PRO-LOG® CORPORATION

STD 7000

7809
6809 CPU Card

USER'S MANUAL

A Pro-Log OPENWARE™ Product

5/85

FOREWORD

This manual explains how to use the Pro-Log 7809. It is structured to reflect the answers to basic questions that you, the user, might ask yourself about the 7809. We welcome your suggestions on how we can improve our instructions.

The 7809 is an OPENWARE product in the Pro-Log Series 7000 STD BUS hardware. Our products are modular, and provide industrial managers with the means of using their own people to control the design, production, and maintenance of the products that use STD BUS hardware. Pro-Log also offers courses on how to design with and use microprocessors and the STD BUS products.

OPENWARE, a trademark of Pro-Log Corporation, means our products are designed to quality standards with readily available, second-sourced, industry standard components, clear and complete specifications, full documentation, and the option to purchase the manufacturing rights and drawings.

You may also find the following Pro-Log documents useful in your work: the **Microprocessor User's Guide** and the **STD BUS Technical Manual and Product Catalog**. If you would like a copy of these documents, please write to us on your company letterhead.

Contents

	Page
Foreword	ii
Figures	v
References and Bibliography	vi
Section 1. Purpose and Main Features	1-1
Purpose	1-1
Features	1-1
The 6809 Processor	1-2
Section 2. Installation and Specifications	2-1
Introduction	2-1
Installation	2-1
Card Layout and Jumpers	2-2
Selecting Memory and I/O Mapping	2-3
Configuring the Memory Sockets (W5-W9)	2-3
Memory and I/O Decoder PROM	2-3
MEMEX and IOEXP (W2, W4)	2-6
I/O Address Comparator (W11)	2-6
Bus Transfer Dead Cycle (W10)	2-7
Interrupts (W13)	2-8
WAITRQ* and BUSRQ* (W3, E1, E2)	2-8
Serial Channel (W16-W20)	2-8
Refresh* (W1)	2-8
Reset Switch (W15)	2-8
Reset Circuit	2-8
Specifications	2-9
Electrical Specifications	2-9
Environmental Specifications	2-9
Timing Specifications	2-9
Mechanical Specifications	2-15
Section 3. Operation	3-1
Introduction	3-1
Asynchronous Communications Interface Adapter (ACIA)	3-1
Transmit and Receive Data Registers	3-2
Read Status Register	3-2
Control Register	3-2
Command Register	3-3
ACIA Reset	3-3
Interrupt Vector Latch	3-3
Section 4. Operating Software	4-1
Introduction	4-1
Memory Addresses	4-1
I/O Addresses	4-1
Description	4-1
Subroutine Specifications:	
(INIT ACIA)	4-2
(STRT XMT)	4-2
(RCV & ECHO)	4-3
(INTERRUPT)	4-3
Section 5. Maintenance	5-1
Reference Drawings	5-1
Return for Repair/Warranty	5-6
Appendix A—The 6809	A-1
The 6809	A-1
6809 Signal Descriptions	A-1
System Organization	A-2
Central Processing Unit	A-2
7809 System	A-2
6809 Architecture	A-3
Instruction Register	A-3
Program Address Counter	A-3
Stack Pointers	A-3
Direct Page Register	A-3
Index Registers	A-3
Data ALU and Accumulators	A-3
Flag Register (Status)	A-3
Memory	A-3
Inputs and Outputs	A-3
Interrupt	A-4
Summary	A-4
Programming Methods	A-4
Position Independence	A-4
Modular Programming	A-4
Re-entrancy/Recursion	A-5
6809 Capabilities	A-5
Position-Independent Code	A-5
Re-entrant Programs	A-6
Recursive Programs	A-6
Loops	A-6
Stack Programming	A-6
Addressing Modes	A-7
Inherent (Includes Accumulator)	A-7
Immediate Addressing	A-7
Direct Addressing	A-7
Direct Indirect	A-8
Short Direct Addressing	A-8
Register Addressing	A-8
Indexed Addressing	A-8
Zero-Offset Indexed	A-9
Constant Offset Indexed	A-9
Accumulator Offset Indexed	A-9

Contents (cont.)

	Page
Auto Increment/Decrement Indexed	A-10
Indexed Indirect	A-10
Relative Addressing	A-10
Program Counter Relative	A-10
Generating the Post Byte	A-11
TFR/EXG	A-11
Transfer/Exchange Post Byte	A-11
LDXE, LDYE, LDUE, LDSE	A-11
Indexed Addressing Post Byte	A-12
6809 Programming Applications	A-13
Accesses to Data on the Stack	A-13
Direct Addressing of Memory Where the Memory Block Moves	A-13
Stepping Through Data or Tables	A-13
Indentical Operations on Tables of Data	A-14
Row-Column Tables	A-14
Determining the Current Value of the Stack	A-15
Storing Without Interruptions	A-15
Calling a User Routine After a Breakpoint	A-15
Saving Working Registers	A-15
The U Register and the User's Stack	A-16
Block Moves	A-16
Cutting Down the Number of Bytes Used	A-16
R Register	A-16
D Register	A-16
Direct Addressing	A-16
Short Relative Addressing	A-16
Automatic Increment and Decrement	A-16
Obtaining Data from an Address on the Stack	A-16
Pulling the Stack and Returning	A-16
Appendix B. The 6809 Instruction Set	B-1
The 6809 Instruction Set	B-1
STD Instruction Mnemonics	B-1

Figures

Figure		Page
1-1	7809 6809 CPU Card	1-1
1-2	Block Diagram	1-2
2-1	Installation	2-1
2-2	Physical Layout and Jumper Options	2-2
2-3	Decoder PROM Part Numbers and Programming Equipment	2-4
2-4	Decoder PROM Program Code	2-4
2-5	Address Map Within the PROM	2-4
2-6	Memory and I/O Decoder PROM Example	2-5
2-7	Memory and I/O Map as Shipped	2-5
2-8	I/O Address Upper 8 Bits	2-6
2-9	I/O Address Lower 8 Bits	2-7
2-10	Onboard I/O Address Selection	2-7
2-11	Electrical and Environmental Specifications	2-9
2-12	Serial Switching Levels	2-9
2-13	Read timing	2-10
2-14	Write Timing	2-11
2-15	INTRQ* and NMIRQ* Interrupt Timing	2-12
2-16	Wait State Timing	2-13
2-17	Clock Stretching	2-14
2-18	Serial Connector Pin List	2-15
2-19	Compatible Connectors	2-15
2-20	STD BUS Edge Connector	2-15
3-1	I/O Port Functions	3-1
3-1	ACIA Internal Organization	3-1
4-1	Subroutine Types	4-1
4-2	Main Subroutine User Requirements	4-2
4-3	1K Program Memory Map	4-4
4-4	256-Byte RAM Memory Map	4-5
4-5	(INIT ACIA) Flowchart	4-6
4-6	(STRT XMT) Flowchart	4-6
4-7	(XMT BLK) Flowchart	4-6
4-8	(XMT CHAR) Flowchart	4-7
4-9	(END XMT?) Flowchart	4-7
4-10	(RCV & ECHO) Flowchart	4-7
4-11	(XMT OFF) Flowchart	4-7
4-12	(INTERRUPT) Flowchart	4-8
4-13	7809 Software Listing	4-9
5-1	Schematic	5-2
5-2	Assembly Drawing	5-5
A-1	6809 Pin Assignments	A-1
A-2	System Data Flow	A-2
A-3	Stacking Order	A-7
A-4	Indexed Addressing Post Byte Register Bit Assignments	A-8
A-5	Indexed Addressing Mode	A-9
A-6	Load Register Effective Examples	A-11
A-7	Indexed Addressing Mode	A-12
B-1	Examples of Instruction Mnemonic Structure	B-1
B-2	6809-STD Mnemonic Equivalency	B-2
B-3	6809-STD Terminology Equivalency	B-2
B-4	Load and Store Instructions	B-3
B-5	Relative Jump, Arithmetic Instructions	B-4
B-6	Program Address Control, Logical Instructions	B-5
B-7	Forward Relative Offset	B-6
B-8	Backward Relative Offset, Special Operations	B-7
B-9	Stacking Order, Hexadecimal and Decimal Conversion	B-8
B-10	ASCII Set, Indexed Post Byte Bit Assignments	B-10
B-11	Reverse Assembly	B-12
B-12	Reverse Assembly, Indexed Addressing Modes	B-13
B-13	Indexed Post Byte Permutations	B-14
B-14	All Permutations of Non-Indirect 5-Bit Offset	B-14
B-15	Operating Notation	B-15
B-16	Register Notation	B-15

REFERENCES/BIBLIOGRAPHY

6809:

MC6809-MC6809E Microprocessor Programming Manual, M6809 PM(AD), Motorola Inc., 1981

MC6809 (Specification Sheet DS9845,) Motorola Inc., 1981.

MC6809 Microprocessor Instruction Set Summary, Motorola Inc., 1978.

Motorola Microprocessors Data Manual, Motorola Inc., 1981.

Microcomputer Data Book, Hitachi, Ltd., #U20.

6551 ACIA:

Rockwell 1984 Data Book, Rockwell International, October 1983, Order No. 1.

Synertek Data Book 1983, Synertek Inc.

1984 MOS PRODUCTS CATALOG, 2nd Edition, Gould/AMI

Section 1
Purpose and Main Features

a highly versatile 6809-based processor
rd memory capability and a serial RS-232-C
is available in two versions: the 7809-1
6 MHz and the 7809-2 operates at 1.8432

ort supports 15 standard asynchronous
n 50 to 19,200 baud for full duplex data
ns equipment (DCE) or data terminal
TE) applications. The serial port can be
r stop bits, parity generation/detection,
ntrol lines.

y consists of five 28-pin JEDEC standard
sockets accept combinations of 2K or 8K
and 2K-, 4K-, or 8K-byte EPROMs.

o features an interrupt vector latch which
ntents of the data bus during interrupt
his register can be read at any time. This
he 7809 to take advantage of peripheral
ate Z80 Mode 2 interrupts.

FEATURES

- 6809 processor
- Two operating versions: 1.8432 MHz (7809-2 with 7.3728 MHz input clock and .9216 MHz (7809-1 with 3.6864 MHz input clock)
- Up to 40K bytes of onboard EPROM capacity or any combination of 2K or 8K RAM or 2K, 4K, or 8K EPROM
- RS-232-C serial port with handshake
- Selectable DCE/DTE configuration
- NMI pushbutton for warm boot or PBRESET switch for system reset
- Direct Memory Access (DMA) to onboard memory capability
- Power-on reset circuit with V_{CC} sensing.

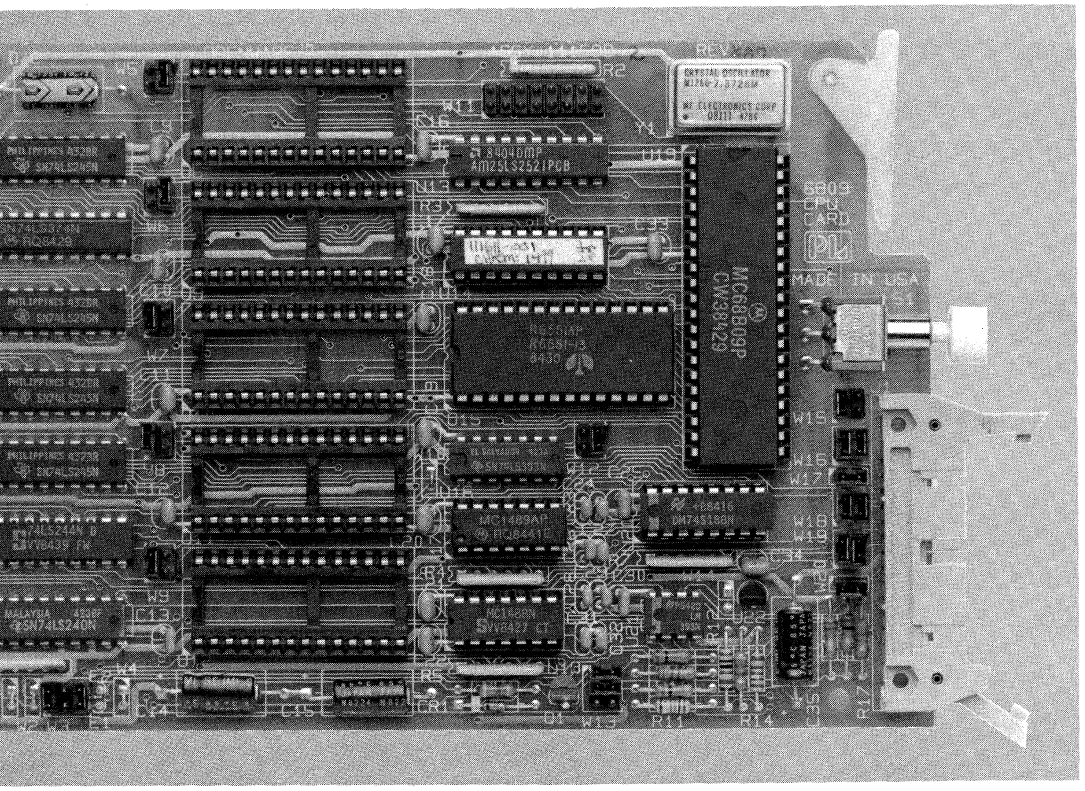


Figure 1-1. 7809 6809 CPU Card

MEMEX and IOEXP (W2, W4)

MEMEX is the memory expansion line of the STD BUS and IOEXP is the I/O expansion line. The 7809 does not use MEMEX or IOEXP and treats these lines as don't care. MEMEX and IOEXP are connected to ground on the 7809 by W2 and W4 respectively. Both signals are pulled up, so they become high if their associated jumpers are removed. Once the associated jumper is removed, either signal can be controlled by another card in the system.

I/O ADDRESS COMPARATOR (W11)

I/O is placed in a 2K address range by the decoder PROM as explained in **Memory and I/O Decoder PROM**, page 2-4. The onboard I/O device addresses are further decoded by the I/O address comparator and W11. On-board I/O is decoded from a 2K block to any 8-word block on a boundary divisible by eight.

The I/O Address Comparator (U13) compares address lines CA10 through CA3 with their associated jumpers at W11. If the associated jumper is removed, the address line is compared to a pullup (high). If the associated jumper is installed, the address line is compared to ground (low). If CIORQ* is active low and all address lines CA10-CA3 match their associated jumpers, then an 8-word on-card I/O access is recognized.

The 16-bit address is decoded into the 2K I/O address range by the memory and I/O decoder PROM (U20). This results in the 2K address range of the form n000-n7FF or n800-nFFF. The remaining upper 8 bits of the I/O address are decoded by W11 jumpers 15-16, 13-14, and 11-12 as shown in figure 2-8.

I/O ADDRESS	n000-n7FF W11 JUMPER			n800-nFFF W11 JUMPER			I/O ADDRESS
	15-16	13-14	11-12	15-16	13-14	11-12	
n0nn	X	X	X	X	X	X	n8nn
n1nn	X	X		X	X		n9nn
n2nn	X		X	X		X	nAnn
n3nn	X			X			nBnn
n4nn		X	X		X	X	nCnn
n5nn		X			X		nDnn
n6nn			X			X	nEnn
n7nn							nFnn

X=Jumper Installed

Figure 2-8. I/O Address Upper 8 Bits

W11 9-10, 7-8, 5-6, 3-4, and 1-2 will further decode the I/O address into 8-word blocks as shown in figure 2-9.

	W11 JUMPERS				
I/O ADDRESS	9-10	7-8	5-6	3-4	1-2
nn00-nn07	X	X	X	X	X
nn08-nn0F	X	X	X	X	
nn10-nn17	X	X	X		X
nn18-nn1F	X	X	X		
nn20-nn27	X	X		X	X
nn28-nn2F	X	X		X	
nn30-nn37	X	X			X
nn38-nn3F	X	X			
nn40-nn47	X		X	X	X
nn48-nn4F	X		X	X	
nn50-nn57	X		X		X
nn58-nn5F	X		X		
nn60-nn67	X			X	X
nn68-nn6F	X			X	
nn70-nn77	X				X
nn78-nn7F	X				
nn80-nn87		X	X	X	X
nn88-nn8F		X	X	X	
nn90-nn97		X	X		X
nn98-nn9F		X	X		
nnA0-nnA7		X		X	X
nnA8-nnAF		X		X	
nnB0-nnB7		X			X
nnB8-nnBF		X			
nnC0-nnC7			X	X	X
nnC8-nnCF			X	X	
nnD0-nnD7			X		X
nnD8-nnDF			X		
nnE0-nnE7				X	X
nnE8-nnEF				X	
nnF0-nnF7					X
nnF8-nnFF					

Figure 2-9. I/O Address Lower 8 Bits

Within this 8-word I/O address block, address line CA2 selects between the ACIA and interrupt vector latch. When CA2 is a one, the ACIA is selected. Address lines CA1 and CA0 select the registers within the ACIA:

CA1	CA0	WRITE	READ
0	0	Transmit Data Register	Receive Data Register
0	1	Programmed Reset	Read Status Register
1	0	Write Command Register	Read Command Register
1	1	Write Control Register	Read Control Register

The interrupt vector latch can be read at any one of the four addresses in its block. The interrupt vector latch address block should not be used by any other I/O device for read or write. This prevents possible bus contention.

Figure 2-10 shows how to select the onboard I/O addresses for the 7809. Figure 3-1, page 3-1, shows the I/O addresses for the card as shipped.

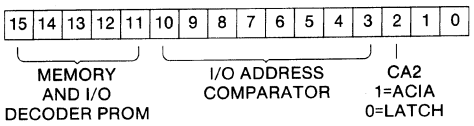


Figure 2-10. Onboard I/O Address Selection

BUS TRANSFER DEAD CYCLE (W10)

The 6809 processor defines a dead cycle that is required during each bus transfer in which the DMA*/BUSRQ* pin is low for more than 14 clock cycles. The processor uses the dead cycle time for self-refresh. The 6809 indicates bus grant when its BA and BS pins are high. On the 7809 as shipped, this results in an immediate generation of the BUSAK* signal. Thus BUSAK* will be low, granting the bus during the first dead cycle of the transfer of control to the requestor. However, the definition of the BUSAK* signal for the STD BUS states that BUSAK* indicates that the bus is available for use by the requestor.

Extensive testing shows it is not necessary to delay the BUSAK* signal by one clock cycle to prevent the signal from being active during a dead cycle. However, should the user require such delay circuitry, the jumper at W10 must be removed to allow the user to add dead cycle delay circuitry.

INTERRUPTS (W13)

The STD BUS has two interrupt lines: NMIRQ* and INTRQ*. The 6809, however, has three interrupt pins. For this reason, the STD INTRQ* line can be jumpered to either the 6809 INTRQ* or FIRQ* pin. The ACIA interrupt has this same interrupt jumpering choice.

The 6809 fast interrupt request pin (FIRQ*) has priority over the interrupt request pin (IRQ*). FIRQ* stacks only the flag register and program counter on an interrupt; this makes it faster than IRQ*, which stacks the entire machine state.

Jumper W13 allows the user to determine the relative speed and priority of the STD INTRQ* line and the ACIA interrupt request. Both of these signals can be connected to the same 6809 interrupt pin. If both signals are to be connected to FIRQ*, the connection must be wire-wrapped. In all other cases, shorting plugs can be used.

W13 CONNECTION	INTERRUPT SOURCE	6809 INTERRUPT PIN
1-2	INTRQ*	IRQ*
3-4	ACIA	IRQ*
5-6	INTRQ*	FIRQ*
3-5	ACIA	FIRQ*

WAITRQ* AND BUSRQ* (W3, E1, E2)

The STD WAITRQ* line can be jumpered by W3 to 6809 HALT* (W3 pins 3-4) or MRDY (W3 pins 1-2). The 7809 connects WAITRQ* to MRDY as shipped. When jumpered to the 6809 MRDY pin, WAITRQ* stretches the E and Q clocks to extend the data access time for slow memories. The maximum clock stretch without the loss of internal processor registers is 10 microseconds.

When jumpered to the 6809 HALT* pin (W3 pins 3-4), WAITRQ* causes the processor to halt at the end of its instruction. The processor can be halted indefinitely without loss of data. While halted, the processor buses are high impedance.

The STD BUSRQ* line is connected to the 6809 HALT* pin by W3 pins 5-6. This jumper can be removed and the STD BUSRQ* line jumpered to 6809 DMA*/BUSRQ* with E1 and E2. However, transitions of DMA*/BUSRQ* should occur during Q (while CNTRL* is active low). This condition may require external circuitry. As was the case for WAITRQ* connected to MRDY, BUSRQ* cannot be held low for more than 10 microseconds when connected to DMA*/BUSRQ* without loss of internal processor registers.

SERIAL CHANNEL (W16-W20)

Jumpers W16-W20 affect the serial channel. Jumper groups W16, W18, and W19 select whether the serial channel operates as data communications equipment (DCE) or data terminal equipment (DTE). When operating the channel as DCE (the as-shipped configuration), W16, W18, and W19 should be jumpered across pins 1-2 and 3-4. To operate the channel as DTE, turn the jumpers sideways to short pins 1-3 and 2-4.

As shipped, the 7809 is configured as DCE. As such, the DCD input is pulled up on-card via W20 pins 1-2, and DCD need not be monitored. W20 pins 2-3 connect the DCD line, which is also known as RLSD, to the ACIA for modem interface. If the jumper is removed, the signal is pulled down on-card and the receiver is disabled.

As shipped, the receiver clock (RxC*) is connected to the serial channel by W17 pins 1-2. RxC* can be used to slave other logic to the ACIA. Removing the W17 jumper leaves the serial channel RxC* line floating.

REFRESH* (W1)

The REFRESH* signal indicates that the processor is not using the signal lines that are required for refreshing dynamic memory as shipped. When REFRESH* is active low, a dynamic RAM refresh can occur. If the dynamic RAM refresh circuitry must control the REFRESH* signal, jumper W1 can be removed to allow REFRESH* to float to be controlled by another card.

RESET SWITCH (W15)

As shipped, the reset switch (S1) is connected by W15 pins 1-2 to NMIRQ*. When pressed, the switch generates a nonmaskable interrupt. This signal can be used for warm boot. The switch is not debounced and can result in multiple interrupts. The warm boot software should take multiple interrupts into account.

The reset switch can also be connected by W15 pins 3-4 to PBRESET*. When the switch is pressed, a system reset occurs. The PBRESET* line has switch debounce circuitry.

RESET CIRCUIT

The reset circuit is designed to allow operation only when the supply voltage is at a valid level. The 7809 reset circuit consists mainly of two voltage comparators. The first comparator does not turn on until the power supply reaches a voltage in the operating range. After the first comparator turns on, indicating a valid operating voltage, reset must be held low until the clock oscillator is functioning. This is accomplished by having the first comparator charge an RC network which forms an input to the

second voltage comparator. The second comparator turns on and sets RST* high after an RC time constant equal to the oscillator startup time.

R13 and U22 generate a reference voltage for the two comparators. R12 and R14 form a voltage divider for the supply voltage. R15 and C35 form the RC time constant network which determines oscillator startup time. R11 and R17 provide hysteresis for the voltage comparators to prevent oscillations.

SPECIFICATIONS

Figures 2-11 through 2-20 give specifications for the 7809 card.

Electrical Specifications

7809 MTBF (calculated using component standard failure rates specified in **Failure Rates and MTBF Calculation**, document #111443) is 22 years.

Environmental Specifications

	RECOMMENDED OPERATING LIMITS			ABSOLUTE NONOPERATING LIMITS		
PARAMETER	MIN	TYP	MAX	MIN	MAX	UNITS
Supply Voltage	4.75	5.00	5.25	0	7.00	V
Supply Current	N/A	860	1420	N/A	N/A	mA
Positive RS-232-C Supply Voltage	11.40	12.00	12.60	0	15.00	V
Positive RS-232-C Supply Current	N/A	19	30	N/A	N/A	mA
Negative RS-232-C Supply Voltage	-11.40	-12.00	-12.60	0	-15.00	V
Negative RS-232-C Supply Current	N/A	21	35	N/A	N/A	mA

	RECOMMENDED OPERATING LIMITS			ABSOLUTE NONOPERATING LIMITS		
PARAMETER	MIN	TYP	MAX	MIN	MAX	UNITS
Free Air Temperature	0	+25	+65	-40	+75	°C
Noncondensing relative humidity	5	N/A	95	5	95	%

Figure 2-11. Electrical and Environmental Specifications (7809)

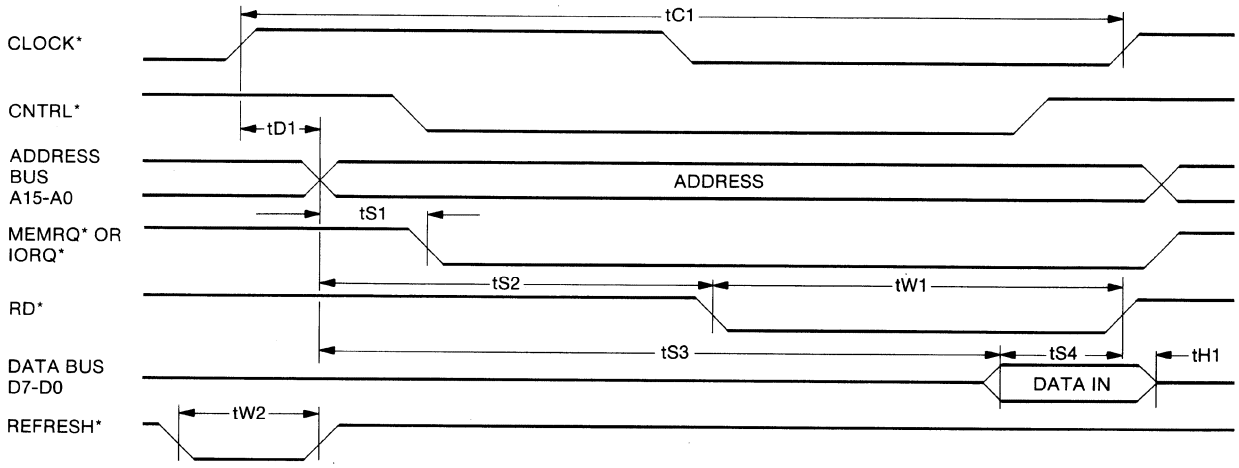
PARAMETER	MIN	TYP	MAX	UNITS
High-level output voltage[1]	9.0	10.5		Volts
Low-level output voltage[1]	-9.0	-10.5		Volts
High-level short-circuit current	6.0	10.0	12.0	mA
Low-level short-circuit current	-6.0	-10.0	-12.0	mA
High-level input threshold voltage			3.0	Volts
Low-level input threshold voltage			-3.0	Volts

[1]At termination resistance of 3.0K minimum

Figure 2-12. Serial Switching Levels (7809)

Timing Specifications

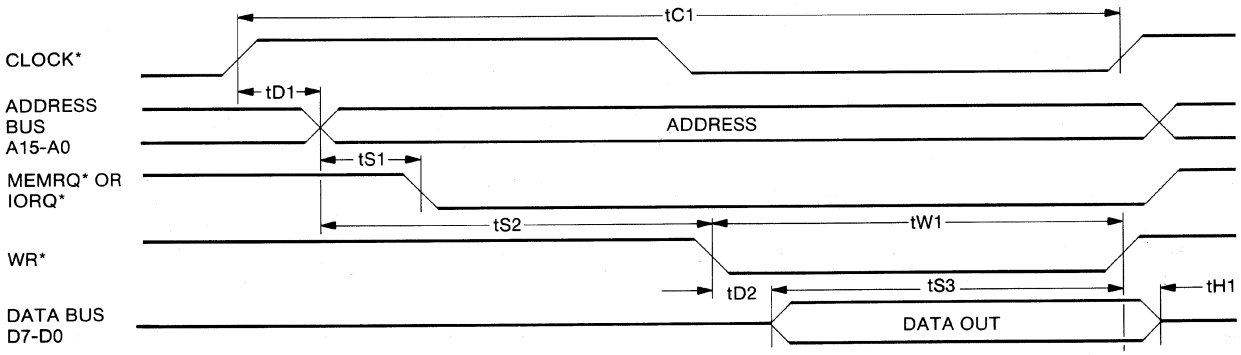
The input to the processor EXTAL pin is four times the frequency of the clock generated at the processor E pin. The two EXTAL input frequencies used by the 7809 are 3.6864 MHz (7809-1) and 7.3728 MHz (7809-2). These frequencies were chosen to support the baud rates of the serial channel. This results in a .9216 MHz (7809-1) or 1.8432 MHz (7809-2) clock. The 6809 is not specified at these frequencies, thus the timing given in figures 2-13 through 2-17 is specified for 1 MHz and 2 MHz. The timing specified in these figures is for reference only. This timing is specified for unterminated buses or buses terminated as recommended by the STDMG STD BUS Specification and Practice (Revision D).



SYMBOL	PARAMETER	2 MHz MIN	2 MHz MAX	1 MHz MIN	1 MHz MAX
tC1	Clock cycle time	500		1000	
tD1	Address delay from CLOCK*		121		211
tH1	Data hold time after RD* inactive	-10		-10	
tS1	Address setup before MEMRQ* or IORQ* active	22		57	
tS2	Address setup before RD* active	101		231	
tS3	Address setup before data valid	299		664	
tS4	Data setup before RD* inactive	111		151	
tW1	RD* pulse width	210		440	
tW2	Refresh pulse width	70		190	

All times given in nanoseconds

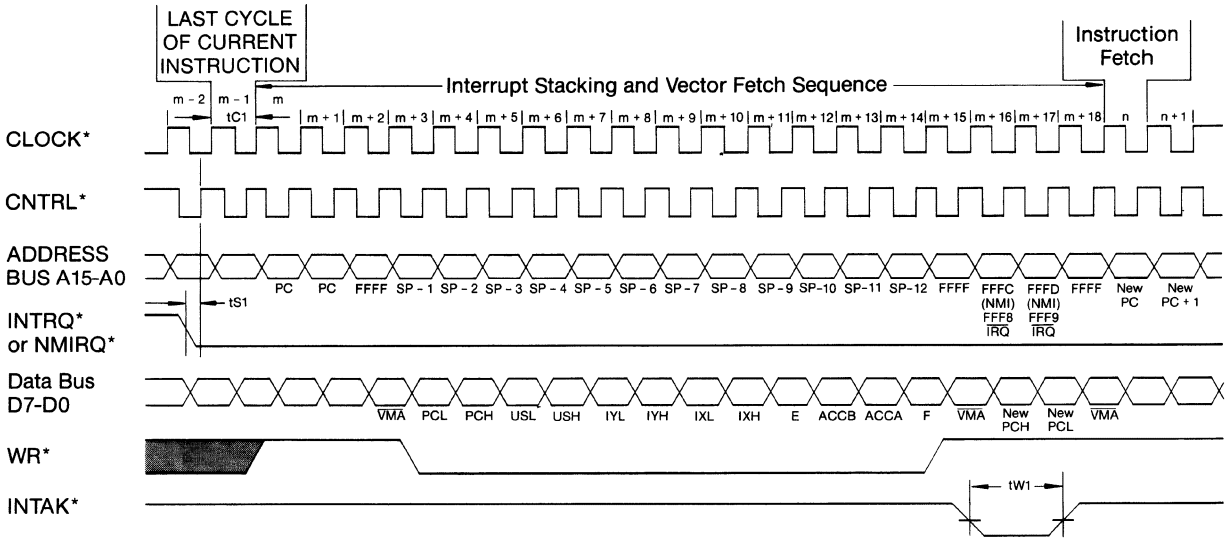
Figure 2-13. Read Timing



		2 MHz		1 MHz	
SYMBOL	PARAMETER	MIN	MAX	MIN	MAX
tC1	Clock cycle time	500		1000	
tD1	Address delay from CLOCK*		121		211
tD2	Data delay from WR* active	57		57	
tH1	Data hold time after WR* inactive	0		0	
tS1	Address setup before MEMRQ* or IORQ* active	22		57	
tS2	Address setup before WR* active	101		231	
tS3	Data setup before WR* inactive	181		411	
tW1	WR* pulse width	210		440	

All times given in nanoseconds

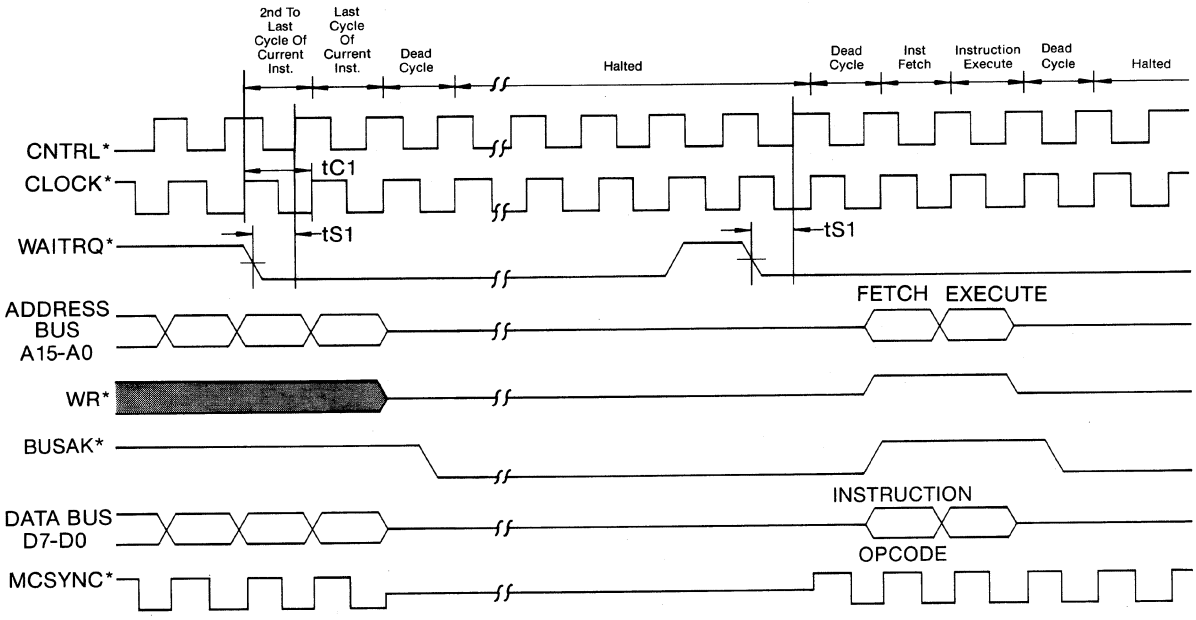
Figure 2-14. Write Timing


LAST CYCLE OF CURRENT INSTRUCTION
Instruction Fetch
Interrupt Stacking and Vector Fetch Sequence
m - 2 m - 1 m
tC1
m + 1 m + 2 m + 3 m + 4 m + 5 m + 6 m + 7 m + 8 m + 9 m + 10 m + 11 m + 12 m + 13 m + 14 m + 15 m + 16 m + 17 m + 18 n n + 1
CLOCK*
CNTRL*
ADDRESS BUS A15-A0
PC PC FFFF SP - 1 SP - 2 SP - 3 SP - 4 SP - 5 SP - 6 SP - 7 SP - 8 SP - 9 SP-10 SP-11 SP-12 FFFF FFFC (NMI) FFF8 IRQ FFFD (NMI) FFF9 IRQ FFFF New PC New PC + 1
tS1
INTRQ* or NMIRQ*
Data Bus D7-D0
VMA PCL PCH USL USH IYL IYH IXL IXH E ACCB ACCA F VMA New PCH New PCL VMA
WR*
tW1
INTAK*

		2 MHz		1 MHz	
SYMBOL	PARAMETER	MIN	MAX	MIN	MAX
tC1	Clock cycle time	500		1000	
tS1	Interrupt setup time	128		218	
tW1	INTAK* pulse width	885		1790	

All times given in nanoseconds

Figure 2-15. INTRQ* and NMIRQ* Interrupt Timing



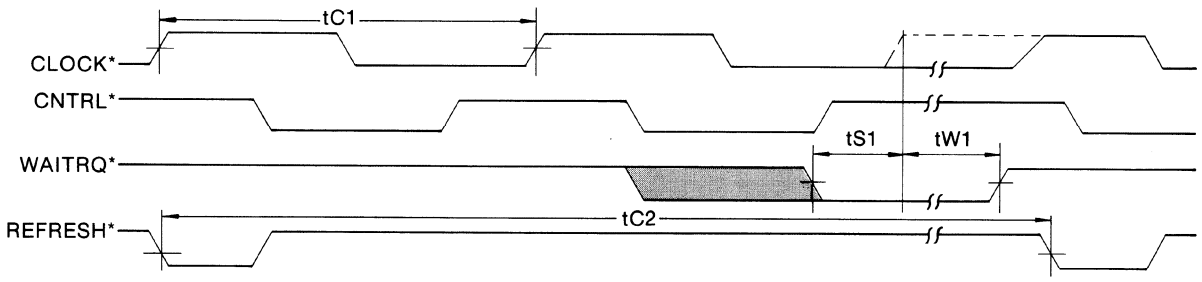
		2 MHz		1 MHz	
SYMBOL	PARAMETER	MIN	MAX	MIN	MAX
tC1	Clock cycle time	500		1000	
tS1	WAITRQ* setup time before CNTRL* clock high	151		241	

All times given in nanoseconds

Figure 2-16. Wait State Timing

A wait state can be substituted for the clock stretching function selected by WAITRQ* as shipped, but the bus will act differently. This card would not be compatible with a card where WAITRQ* is connected to MRDY on the 6809 (but only where WAITRQ* is concerned).

WAITRQ* or BUSRQ* can be used to drive the 6809 HALT* line depending upon the position of W3. See figure 2-2.



		2 MHz		1 MHz	
SYMBOL	PARAMETER	MIN	MAX	MIN	MAX
tC1	Clock cycle time	500	10000	1000	10000
tC2	Refresh cycle time	500	10000	1000	10000
tS1	Pulse stretch setup time	128		218	
tW1	Pulse stretch width		15605		15195

All times given in nanoseconds

Figure 2-17. Clock Stretching

A wait state can be substituted for the clock stretching function selected by WAITRQ* as shipped, but the bus will act differently. This card would not be compatible with a card where WAITRQ* is connected to MRDY on the 6809 (but only where WAITRQ* is concerned). See figure 2-2.

Mechanical Specifications

The 7809 meets all general mechanical specifications of the STD BUS.

PIN # OF SERIAL CONNECTOR J1	SIGNAL NAME	RS-232-C PIN
1	N/C	1
2	N/C	14
3	TD*	2
4	N/C	15
5	RD*	3
6	N/C	16
7	RTS	4
8	N/C	17
9	CTS	5
10	N/C	18
11	DSR	6
12	N/C	19
13	GND	7
14	DTR	20
15	DCD (RLSD)	8
16	N/C	21
17	N/C	9
18	RI	22
19	N/C	10
20	N/C	23
21	N/C	11
22	N/C	24
23	N/C	12
24	N/C	25
25	N/C	13
26	N/C	N/C

NOTE: Pro-Log cables RC701 and RC702 can be used to convert from the serial port connector to a D connector with the standard RS-232-C pinout.

Figure 2-18. Serial Connector Pin List (7809)

MANUFACTURER	26-PIN TRANSITION CONNECTOR
T&B/Ansley	609-2630-1
Berg	66900-226
3M	3399-6026
MANUFACTURER	**25-PIN MALE D CONNECTOR**
Robinson-Nugent	IDD-25P-TG
Winchester	49-1125P
T&B/Ansley	609-25P
MANUFACTURER	**25-PIN FEMALE D CONNECTOR**
Robinson-Nugent	IDD-25S-TG
Winchester	49-1125S
T&B/Ansley	609-25S

NOTE: This list is for customer convenience. Many other manufacturers' connectors are compatible. For more information, consult manufacturer's catalogs

Figure 2-19. Compatible Connectors (7809)

PIN NUMBER				PIN NUMBER			
OUTPUT (LSTTL DRIVE)				OUTPUT (LSTTL DRIVE)			
INPUT (LSTTL LOADS)						INPUT (LSTTL LOADS)	
MNEMONIC							MNEMONIC
+5V	IN		2	1		IN	+5V
GROUND	IN		4	3		IN	GROUND
-5V			6	5			-5V
D7	2	55	8	7	55	2	D3
D6	2	55	10	9	55	2	D2
D5	2	55	12	11	55	2	D1
D4	2	55	14	13	55	2	D0
A15	1	55	16	15	55	1	A7
A14	1	55	18	17	55	1	A6
A13	1	55	20	19	55	1	A5
A12	1	55	22	21	55	1	A4
A11	1	55	24	23	55	1	A3
A10	1	55	26	25	55	1	A2
A9	1	55	28	27	55	1	A1
A8	1	55	30	29	55	1	A0
RD*	1	55	32	31	55	1	WR*
MEMRQ*	1	55	34	33	55	1	IORQ*
MEMEX (GROUND)			36	35	OUT		IOEXP (GROUND)
MCSYNC*		55	38	37	55		REFRESH*
STATUS 0*		55	40	39	55		STATUS 1*
BUSRQ*	5		42	41	55		BUSAK*
INTRQ*	5		44	43	55		INTAK*
NMIRQ*	5		46	45		5	WAITRQ*
PBRESET*	(1)		48	47	55		SYSRESET*
CNTRL*		55	50	49	55		CLOCK*
PCI	IN		52	51	OUT		PCO
AUX GND	IN		54	53		IN	AUX GND
AUX -V	IN		56	55		IN	AUX +V

* Low-level active logic
(1) Switch input

Figure 2-20. STD BUS Edge Connector (7809)

Section 3
Operation

INTRODUCTION

This section covers the operation of the 7809 serial port and interrupt vector latch. These functions are controlled through I/O ports. Figure 3-1 shows the functions of the ports on the 7809.

AS SHIPPED I/O PORT ADDRESS	OUTPUT FUNCTION	INPUT FUNCTION
C7F8-C7FB	Not Used	Interrupt Vector Latch
C7FC	ACIA Transmit Data Register	ACIA Receive Data Register
C7FD	ACIA Programmed Reset	ACIA Read Status Register
C7FE	ACIA Write Command Register	ACIA Read Command Register
C7FF	ACIA Write Control Register	ACIA Read Control Register

Figure 3-1. I/O Port Functions (7809)

ASYNCHRONOUS COMMUNICATIONS INTERFACE ADAPTER (ACIA)

Operating the serial port is mainly a matter of using the internal registers of the control chip (ACIA). Figure 3-2 shows the internal organization of the ACIA.

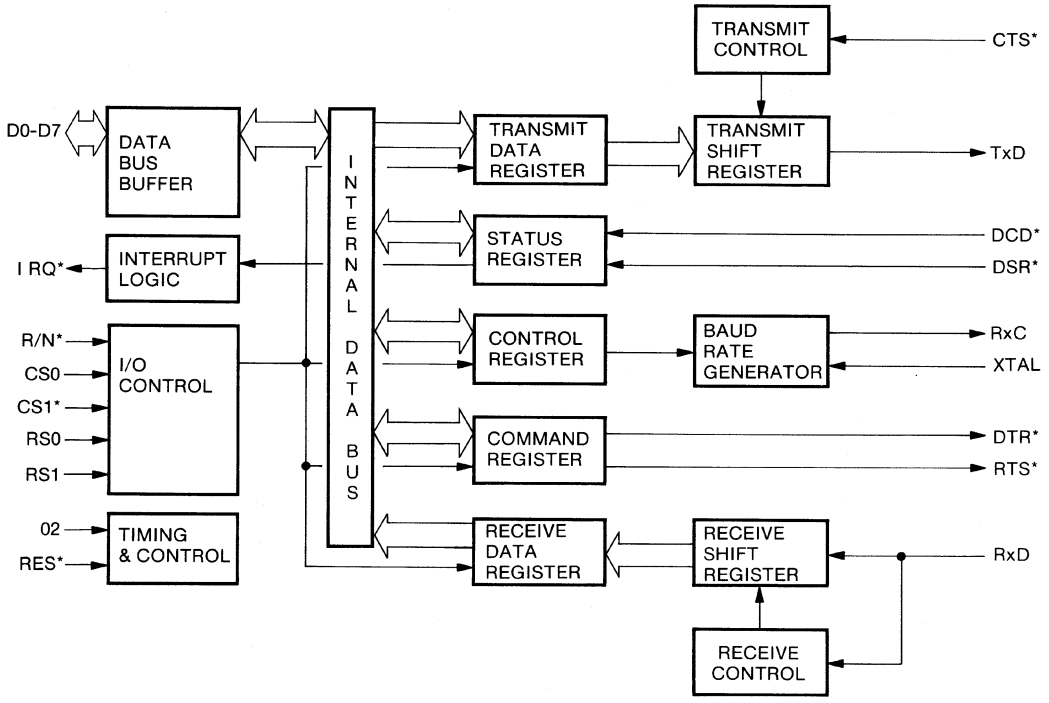


Figure 3-2. ACIA Internal Organization (7809)

Transmit and Receive Data Registers

These registers provide temporary storage for the ACIA transmit and receive circuits. Bit 0 (the least significant bit) is the first bit to be transmitted or received. Unused bits are don't care in the transmitter and zero in the receiver. Parity bits are stripped off before data is stored in the Receive Data Register.

Read Status Register

This register indicates the status of various ACIA functions. Data Set Ready, Data Carrier Detect, Transmit Data Register Empty, and Receive Data Register Full are represented by bits 6 through 3 respectively. If any of these bits are set, the interrupt indicator (bit 7) will also be set.

D7	D6	D5	D4	D3	D2	D1	D0
IRQ	DSR*	DCD*	TDRE	RDRF	OVRN	FE	PE

Interrupt, IRQ (D7): Set to one for an interrupt. This bit is set whenever one or more of bits 6 through 3 are set. D7 is cleared by a read of the register. Bit 1 of the Command Register (IRD) disables interrupts if set to 1.

Data Set Ready, DSR* (D6): Set to one for DSR* high (not ready). D6 reflects the state of the DSR* pin. Whenever bit D6 (DSR*) or D5 (DCD*) change state, an immediate interrupt (IRQ) occurs (unless inhibited by Command Register bit 1, IRD). When the interrupt occurs, the status bits indicate the status of the DSR* and DCD* inputs immediately after the change of state occurred. Subsequent level changes will not affect the status bits until the Status Register is read. At that time, another interrupt immediately occurs and the status bits reflect the new input levels. These bits are not cleared by reset.

When DCD* is high (not detected), the ACIA receiver is disabled.

Data Carrier Detect, DCD* (D5): Set to one for DCD* high (not detected) D5 reflects the state of the DCD* pin. Whenever bit D6 (DSR*) or D5 (DCD*) change state, an immediate interrupt (IRQ) occurs (unless inhibited by Command Register bit 1, IRD). When the interrupt occurs, the status bits indicate the status of the DSR* and DCD* inputs immediately after the change of state occurred. Subsequent level changes will not affect the status bits until the Status Register is read. At that time, another interrupt immediately occurs and the status bits reflect the new input levels. These bits are not cleared by reset.

Transmit Data Register Empty, TDRE (D4): Set to one for empty. This occurs when the ACIA transfers data from the Transmit Data Register to the Transmit Shift Register. D4 is cleared by new data written to the Transmit Data Register.

Receive Data Register Full, RDRF (D3): Set to one for full. This occurs when the ACIA transfers data from the Receive Shift Register to the Receive Data Register. D3 is cleared by data read from the Receive Data Register.

Overrun, OVRN (D2): Set to one for an overrun error. D2 does not cause an interrupt, so it is normally checked when the Receive Data Register is read. D2 is automatically cleared by a read of the Receive Data Register.

Framing Error, FE (D1): Set to one for a framing error. D1 does not cause an interrupt, so it is normally checked when the Receive Data Register is read. D1 is cleared by a read of the Receive Data Register.

Parity Error, PE (D0): Set to one for a parity error. D0 does not cause an interrupt, so it is normally checked when the Receive Data Register is read. D0 is cleared by a read of the Receive Data Register.

Control Register

The Control Register is used to select baud rate, clock control, word length, and the number of stop bits.

D7	D6	D5	D4	D3	D2	D1	D0
SBN	WL		RCS		SBR		

Stop Bit Number, SBN (D7): This bit selects the number of stop bits to be used. A zero indicates one stop bit; a one indicates one and a half stop bits if the word length is 5 with no parity selected, one stop bit if the word length is 8 with parity selected, or two stop bits in all other cases.

Word Length, WL (D6-D5): Bits D6 and D5 determine the number of bits in a word:

BIT		DATA WORD LENGTH
6	5	
0	0	8
0	1	7
1	0	6
1	1	5

Receiver Clock Source, RCS (D4): This bit controls the clock source for the Receiver. A zero causes the Receiver to operate from an external clock that has not been implemented on the 7809. A one causes the Receiver to operate at the same baud rate as that selected for the Transmitter.

Selected Baud Rate, SBR (D3-D0): These bits select the Transmitter baud rate. For the 7809, the Receiver must use the same baud rate as the Transmitter, so RxC becomes an output. RxC can be used to slave other logic to the ACIA.

BIT 3 2 1 0	SELECTED BAUD RATE
0 0 0 0	Unused on 7809
0 0 0 1	50
0 0 1 0	75
0 0 1 1	109.92
0 1 0 0	134.58
0 1 0 1	150
0 1 1 0	300
0 1 1 1	600
1 0 0 0	1200
1 0 0 1	1800
1 0 1 0	2400
1 0 1 1	3600
1 1 0 0	4800
1 1 0 1	7200
1 1 1 0	9600
1 1 1 1	19,200

Command Register

The Command Register controls transmit and receive modes and functions.

D7	D6	D5	D4	D3	D2	D1	D0
PMC		PME	REM	TIC		IRD	DTR

Parity Mode Control, PMC (D7-D6): These bits determine the type of parity generated by the Transmitter and the type of parity check done by the Receiver. A mark parity is a parity bit that is always one; space parity is always zero.

BIT 7	BIT 6	PARITY OPERATION
0	0	Odd parity transmitted and received
0	1	Even parity transmitted and received
1	0	Mark parity transmitted Parity check disabled
1	1	Space parity transmitted Parity check disabled

Parity Mode Enabled, PME (D5): When set to one, this bit enables parity bit generation by the Transmitter and parity bit checking by the Receiver. A zero disables parity bit generation and checking.

Receiver Echo Mode, REM (D4): When set to one, this bit enables the Receiver Echo mode. When D4 is set to one, D3 and D2 must be set to zero. In the Receiver Echo mode, the Transmitter returns each transmission received by the Receiver delayed by one-half bit time. A zero disables the Receiver Echo mode.

Transmitter Interrupt Control, TIC (D3-D2): These bits control the state of the RTS* line and the Transmitter interrupt. These bits can also cause a break to be transmitted.

BIT 3	BIT 2	TRANSMITTER INTERRUPT	RTS* LEVEL	TRANSMITTER
0	0	Disabled	High	Off
0	1	Enabled	Low	On
1	0	Disabled	Low	On
1	1	Disabled	Low	Transmit Break

Receiver Interrupt Control, IRD (D1): When set to one, this bit disables the Receiver from generating an interrupt. The receiver interrupt at bit 3 of the Status Register is enabled when this bit is set to a zero and bit 0 is set to a one.

Data Terminal Ready, DTR (D0): This bit enables all selected interrupts and controls the state of the DTR* line. A zero sets DTR* high and disables the Receiver and all interrupts. A one sets DTR* low and enables the Receiver and all interrupts.

ACIA RESET

The ACIA can be reset in two different ways: by hardware and programmed reset. The two types of reset have different effects on the ACIA registers as shown by the following:

	7 6 5 4 3 2 1 0	
STATUS REGISTER	0 — — 1 0 0 0 0	Hardware Reset
	— — — — — 0 — —	Program Reset
CONTROL REGISTER	0 0 0 0 0 0 0 0	Hardware Reset
	— — — — — — — —	Program Reset
COMMAND REGISTER	0 0 0 0 0 0 0 0	Hardware Reset
	— — — 0 0 0 0 0	Program Reset

INTERRUPT VECTOR LATCH

The interrupt vector latch is intended to take advantage of any cards in the system that generate Z80 Mode 2 interrupts. The interrupt vector latch (U12) latches the contents of the data bus on the high going edge of INTAK*. This captures the Mode 2 interrupt vector that

identifies the interruptor. The contents of this latch can be examined at any time by an I/O read. As shipped, the interrupt vector latch is located at hex addresses C7F8-C7FB. The latch can be read by accessing any one of its four addresses. The contents of the interrupt vector latch can be used in selecting the correct interrupt service routine.

Section 4
Operating Software

INTRODUCTION

This section contains hardware-level subroutine modules with which to operate the 7809 card. The software in this section can be used without license from Pro-Log. Although tested and believed correct, this software is not represented to be free from errors or copyright infringement, or appropriate for any specific application.

The subroutines are in STD instruction mnemonics. The coding forms are grouped at the end of this section following the flowcharts and subroutine specifications.

MEMORY ADDRESSES

Full memory addresses are given. They are preferred addresses that allow the subroutines to work with those provided for other Series 7000 STD BUS cards from Pro-Log. The program addresses correspond to the 7809 onboard memory sockets.

If your system cannot use the memory addresses in the 7809 software package, simply change the memory page addresses, as required, when loading these modules into your system.

I/O PORT ADDRESSES

The 7809 I/O ports are assigned preferred hexadecimal addresses C7F8-C7FF for compatibility with other Series 7000 cards. Section 2 explains how to remap these addresses if necessary. This software can be used by simply changing the port address when loading the program modules into your system.

DESCRIPTION

There are eight subroutines, many of which are subroutines within subroutines that do not need to be accessed directly. Only four of the subroutines need to be accessed directly to perform serial communication. The remainder could be useful as building blocks by the user's own software for data communication. All of the subroutines are shown in figure 4-1.

NAME	DESCRIPTION
(INIT ACIA)	Initialize ACIA
(STRT XMT)	Start serial block transmission
(END XMT?)	Check if entire block has been transmitted
(XMT BLK)	Transmit a block of data
(XMT CHAR)	Transmit a character
(XMT OFF)	Turn off transmitter interrupt at end of block
(RCV & ECHO)	Enable the receiving and echoing of characters
(INTERRUPT)	Process transmit buffer empty or receive register full interrupts

Figure 4-1. Subroutine Types (7809)

Responses to errors in data reception or changes in incoming serial handshake lines are largely application dependent; therefore, there is no software provided for these and similar operations. The user needs to write the application-dependent software. However, if used as a package, this software greatly simplifies the use of the serial port.

If used as a package accessing just the four main subroutines, only a few requirements must be met by the user. The four main subroutines and their requirements are listed in figure 4-2. The following list details the requirements and points to consider before implementing the software.

- The user may wish to provide software for serial handshake control or for responding to changes in incoming serial handshaking.
- Because it is application dependent, no software is provided for response to errors in received data.
- The user may wish to provide a subroutine to store and/or respond to the received data. The data received is simply echoed and room is provided for a Jump to Subroutine Instruction to jump to the user's own subroutine (PROCESS CHAR).
- The baud rate and various other factors selected by the (INIT ACIA) subroutine should be checked to see that they match the application. The initialized values are given in the description of the (INIT ACIA) routine.

NAME	DESCRIPTION	USE REQUIREMENTS
(INIT ACIA)	Initialize ACIA for serial communications	None
(STRT XMT)	Transmit a block of data	Data must be in memory, data pointer in register X, and number of bytes in the block in register A. Initial value stored in the command register must be at F809, which is automatic if (INIT ACIA) is used.
(RCV & ECHO)	Enables interrupts and enables the receiver echo mode	Initial value stored in the command register must be at F809, which is automatic if (INIT ACIA) is used.
(INTERRUPT)	Transmits the next character when the transmitter is empty or receives a character when the receiver is full	(INTERRUPT) address in 6809 vector table for ACIA interrupts (IRQ, FFF8, as shipped).

Figure 4-2. Main Subroutine Use Requirements

SUBROUTINE SPECIFICATIONS: (INIT ACIA)

ENTRY ADDRESS : F800
MEMORY REQUIREMENTS
  Program Memory : 23
  RAM : 1
  Stack : 0
REGISTERS ALTERED : A, F
SUBROUTINES USED : None
ENTRY REQUIREMENTS : None
EXIT CONDITIONS : ACIA has been program reset and character count storage at 0000 has been initialized to zero.

FUNCTIONAL DESCRIPTION : The subroutine initializes the ACIA and enables the 6809 interrupts. The ACIA is initialized for 2 stop bits, 7 bits/character, and 9600 baud. Even parity is enabled with no receiver echo. RTS* and DTR are set to low. IRQ is enabled, but transmitter interrupt is disabled. These initialized conditions should be checked to make sure they match the application. (INIT ACIA) initializes the character count storage at 0000 to zero for the (END XMT?) subroutine that is called by the (STRT XMT) subroutine.

SUBROUTINE SPECIFICATIONS: (STRT XMT)

ENTRY ADDRESS : F820
MEMORY REQUIREMENTS
  Program Memory : 26 (95 with subroutines)
  RAM : 3
  Stack : 10
REGISTERS ALTERED : None
SUBROUTINES USED : (END XMT?), (XMT BLK), (XMT CHAR), (XMT OFF)
ENTRY REQUIREMENTS :
  Register A : Number of bytes to be sent (FF=255)
  Register X : Address of first byte to be sent
  Memory : Data to be sent
  : 0000 contains the number of bytes remaining in the block currently being transmitted (if any). F809 contains the initial value stored at the command register.
EXIT CONDITIONS : Byte count decremented and stored at 0000, data pointer incremented and stored at 0001 and 0002

FUNCTIONAL DESCRIPTION : This subroutine transmits a block of data of up to 255 bytes. The program first checks to see if a previous block is still in the process of being sent. If it is, the program stays in a continuous checking loop. When the block can be sent, the first byte is sent and the program exits. Thereafter, whenever the transmitter indicates it is ready, via interrupt, the program responds by sending another byte. When the transmitter interrupts after sending the last byte, the program turns off the transmitter's interrupt.

SUBROUTINE SPECIFICATIONS: (RCV & ECHO)

ENTRY ADDRESS : F880
MEMORY REQUIREMENTS
  Program Memory : 17
  RAM : 0
  Stack : 0
REGISTERS ALTERED : A, F
SUBROUTINES USED : None
ENTRY REQUIREMENTS :
  Memory : F809 contains the initial value stored at the command register.
EXIT CONDITIONS : The receiver echo mode is enabled.

FUNCTIONAL DESCRIPTION : This subroutine enables the receiver echo mode and returns. When a character is received, a receiver full interrupt is generated which can be handled by the (INTERRUPT) subroutine.

SUBROUTINE SPECIFICATIONS: (INTERRUPT)

ENTRY ADDRESS : F8A0
MEMORY REQUIREMENTS
  Program Memory : 33 (87 with (XMT BLK) subroutine)
  RAM : 3
  Stack : 13
REGISTERS ALTERED : None
SUBROUTINES USED : (XMT BLK), (PROCESS CHAR) (PROCESS CHAR) must be supplied by the user
ENTRY REQUIREMENTS : None
EXIT CONDITIONS : N/A

FUNCTIONAL DESCRIPTION : This is an interrupt service routine. It checks the ACIA IRQ bit to determine if the interrupt was generated by the ACIA. If the interrupt did not originate from the ACIA, the routine is exited without any action being taken. The routine next checks the ACIA to see if the interrupt was the result of the transmit data register being empty or the receive data register being full. If neither of these is the case, the routine is exited without any action being taken.

If the transmit data register is empty, the routine transmits the next character in the block being transmitted. If the receive data register is full, the routine receives the character. The ACIA automatically echos the character back. The user must supply a subroutine (PROCESS CHAR) to interpret and/or store the received character. A Jump to Subroutine instruction may be placed at address F8BC hex for this purpose.

The receiver is enabled at initialization and, since reception is interrupt driven, the (PROCESS CHAR) routine must flag the processor that a message has been received. Otherwise, data reception is completely transparent to the processor.

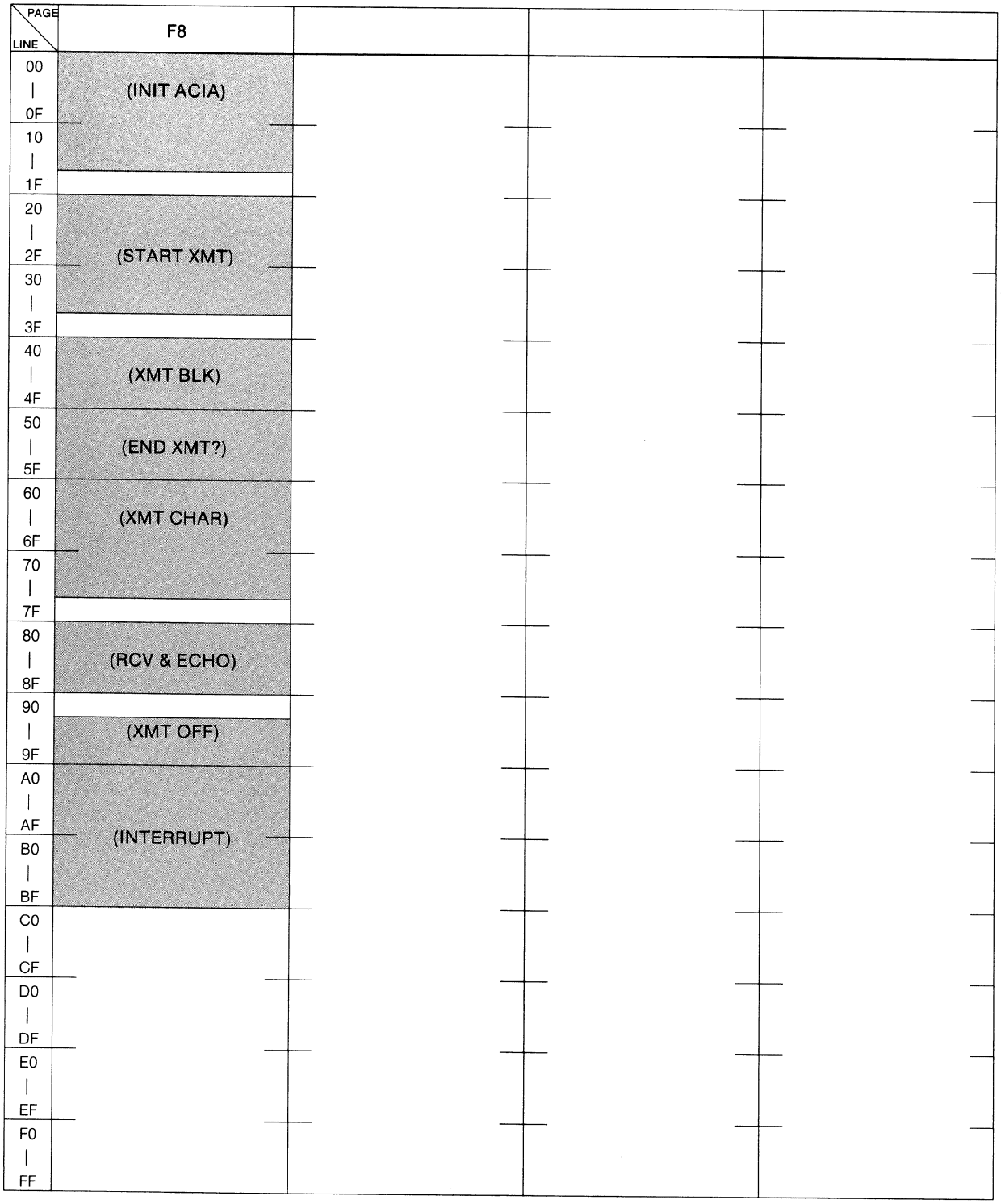


Figure 4-3. 1K Program Memory Map (7809)

PAGE ADDRESS: 00

LINE	LABEL	LINE	LABEL	LINE	LABEL	LINE	LABEL
00	BYTE COUNT	40		80		C0	
01	DATA MSB	41		81		C1	
02	POINTER LSB	42		82		C2	
03		43		83		C3	
04		44		84		C4	
05		45		85		C5	
06		46		86		C6	
07		47		87		C7	
08		48		88		C8	
09		49		89		C9	
0A		4A		8A		CA	
0B		4B		8B		CB	
0C		4C		8C		CC	
0D		4D		8D		CD	
0E		4E		8E		CE	
0F		4F		8F		CF	
10		50		90		D0	
11		51		91		D1	
12		52		92		D2	
13		53		93		D3	
14		54		94		D4	
15		55		95		D5	
16		56		96		D6	
17		57		97		D7	
18		58		98		D8	
19		59		99		D9	
1A		5A		9A		DA	
1B		5B		9B		DB	
1C		5C		9C		DC	
1D		5D		9D		DD	
1E		5E		9E		DE	
1F		5F		9F		DF	
20		60		A0		E0	
21		61		A1		E1	
22		62		A2		E2	
23		63		A3		E3	
24		64		A4		E4	
25		65		A5		E5	
26		66		A6		E6	
27		67		A7		E7	
28		68		A8		E8	
29		69		A9		E9	
2A		6A		AA		EA	
2B		6B		AB		EB	
2C		6C		AC		EC	
2D		6D		AD		ED	
2E		6E		AE		EE	
2F		6F		AF		EF	
30		70		B0		F0	
31		71		B1		F1	
32		72		B2		F2	
33		73		B3		F3	
34		74		B4		F4	
35		75		B5		F5	
36		76		B6		F6	
37		77		B7		F7	
38		78		B8		F8	
39		79		B9		F9	
3A		7A		BA		FA	
3B		7B		BB		FB	
3C		7C		BC		FC	
3D		7D		BD		FD	
3E		7E		BE		FE	
3F		7F		BF		FF	

Figure 4-4. 256-Byte RAM Memory Map (7809)

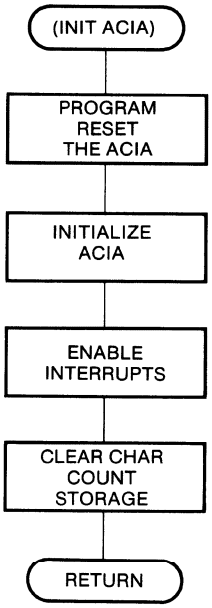


Figure 4-5. (INIT ACIA) Flowchart (7809)

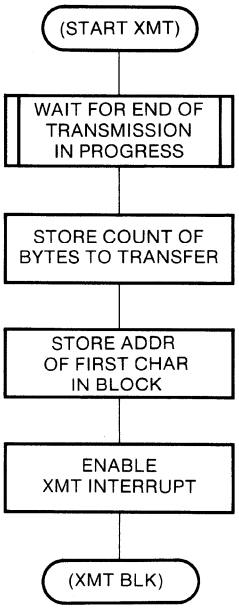

(START XMT)
WAIT FOR END OF TRANSMISSION IN PROGRESS
STORE COUNT OF BYTES TO TRANSFER
STORE ADDR OF FIRST CHAR IN BLOCK
ENABLE XMT INTERRUPT
(XMT BLK)

Figure 4-6. (STRT XMT) Flowchart (7809)

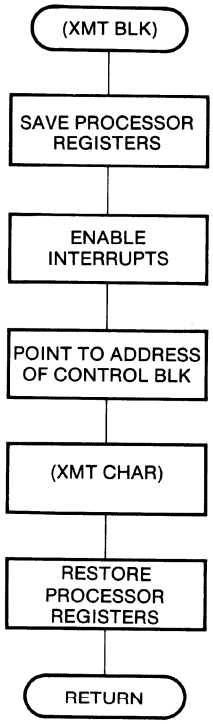


Figure 4-7. (XMT BLK) Flowchart (7809)

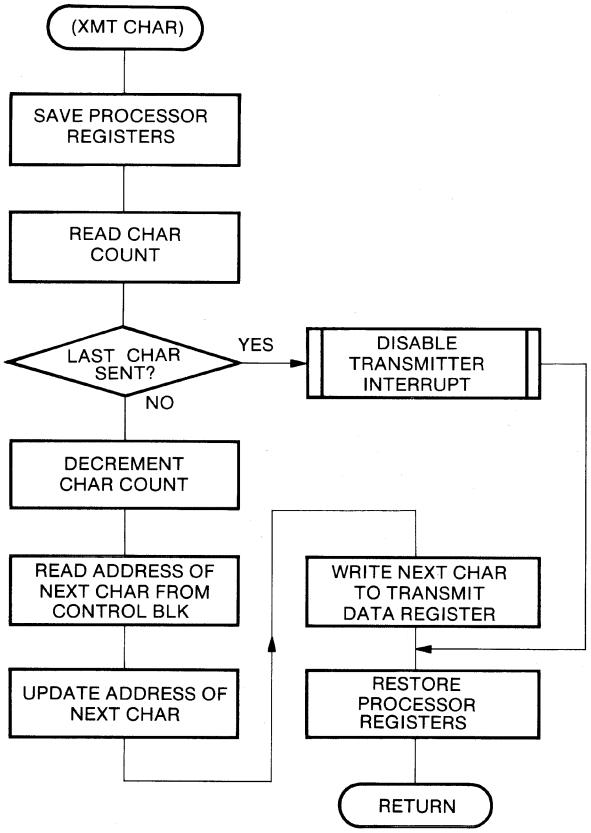


Figure 4-8. (XMT CHAR) Flowchart (7809)

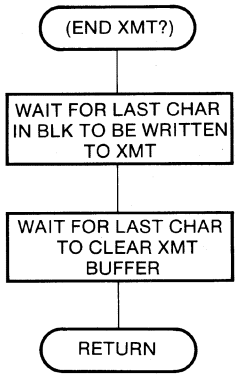


Figure 4-9. (END XMT?) Flowchart (7809)

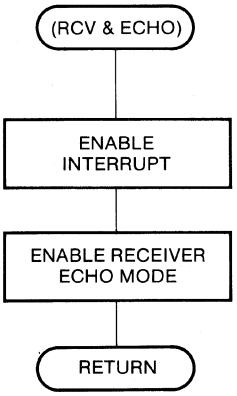


Figure 4-10. (RCV & ECHO) Flowchart (7809)

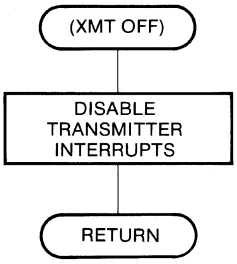


Figure 4-11. (XMT OFF) Flowchart (7809)

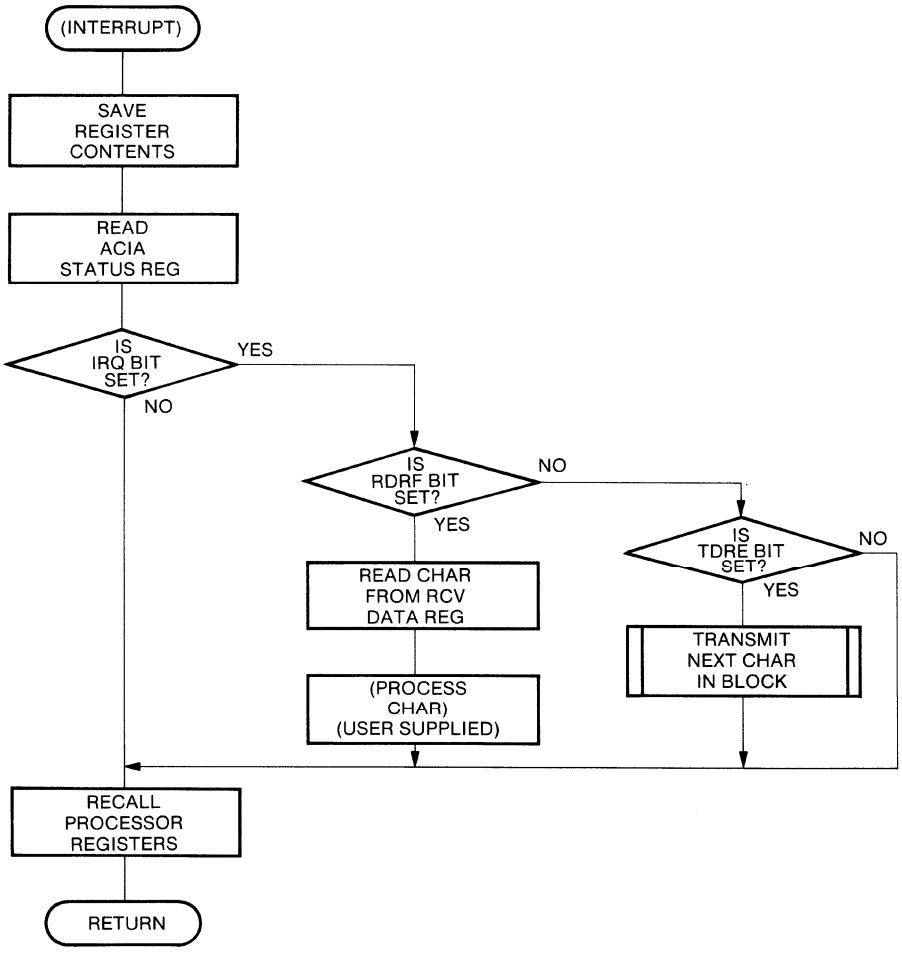


Figure 4-12. (INTERRUPT) Flowchart (7809)

```
ADRdata  |    LABEL     | INSTR |   MODIFIER   |
---------|--------------|-------|--------------|-------------------------------------------------
F800 B7  | (INIT ACIA)  | STAD  |              | PROGRAM RESET THE ACIA
  01 C7  |              |       |     C7FD     |
  02 FD  |              |       |              |
  03 86  |              | LDAI  |              | 2 STOP BITS, 7 BITS/CHAR, 9600 BAUD
  04 BE  |              |       |      BE      |
  05 B7  |              | STAD  |              | STORE AT ACIA CONTROL REGISTER
  06 C7  |              |       |     C7FF     |
  07 FF  |              |       |              |
  08 86  |              | LDAI  |              | EVEN PARITY, PARITY ENABLED, NO ECHO, RTS* LOW,
  09 69  |              |       |      69      |   XMT INT DISABLED, IRQ ENABLED, DTR LOW
  0A B7  |              | STAD  |              |
  0B C7  |              |       |     C7FE     | STORE AT COMMAND REGISTER
  0C FE  |              |       |              |
  0D 1F  |              | LDA   |      F       | GET FLAG REGISTER
  0E A8  |              |       |              |
  0F 84  |              | ANAI  |              | ENABLE INTERRUPTS
F810 EF  |              |       |      EF      |
  11 1F  |              | LDF   |      A       | STORE FLAG REGISTER
  12 8A  |              |       |              |
  13 7F  |              | CLMD  |              | CLEAR CHARACTER COUNT STORAGE FOR FIRST
  14 00  |              |       |     0000     |   (XMT CHAR) CALL
  15 00  |              |       |              |
  16 39  |              | RTS   |              |
  17     |              |       |              |
  18     |              |       |              |
  19     |              |       |              |
  1A     |              |       |              |
  1B     |              |       |              |
  1C     |              |       |              |
  1D     |              |       |              |
  1E     |              |       |              |
  1F     |              |       |              |
F820 34  | (START XMT)  | PSRS  |      A       | SAVE REGISTER CONTENTS
  21 02  |              |       |              |
  22 BD  |              | JS    |      UN      | WAIT FOR END OF TRANSMISSION IN PROGRESS
  23 F8  |              |       |  (END XMT?)  |
  24 51  |              |       |              |
  25 35  |              | PLRS  |      A       | RESTORE REGISTER CONTENTS
  26 02  |              |       |              |
  27 B7  |              | STAD  |              | STORE COUNT OF BYTES TO TRANSFER
  28 00  |              |       |     0000     |
  29 00  |              |       |              |
  2A BF  |              | STXD  |              | STORE ADDRESS OF FIRST CHARACTER IN BLOCK
  2B 00  |              |       |     0001     |
  2C 01  |              |       |              |
  2D B6  |              | LDAD  |              | GET INITIAL VALUE STORED AT COMMAND REGISTER
  2E F8  |              |       |     F809     |
  2F 09  |              |       |              |
F830 84  |              | ANAI  |              | CLEAR TRANSMITTER INTERRUPT CONTROL BITS
  31 F3  |              |       |      F3      |
  32 8A  |              | ORAI  |              | ENABLE TRANSMITTER INTERRUPT, RTS* LOW
  33 04  |              |       |      04      |
  34 B7  |              | STAD  |              | STORE AT COMMAND REGISTER
  35 C7  |              |       |     C7FE     |
  36 FE  |              |       |              |
  37 7E  |              | JP    |      UN      | SEND FIRST CHARACTER IN BLOCK
  38 F8  |              |       |  (XMT BLK)   |
  39 40  |              |       |              |
  3A     |              |       |              |
  3B     |              |       |              |
  3C     |              |       |              |
  3D     |              |       |              |
  3E     |              |       |              |
  3F     |              |       |              |
```

Figure 4-13. 7809 Software Listing

```
ADRdata  |    LABEL     | INSTR  |   MODIFIER    |
---------|--------------|--------|---------------|-------------------------------------------------------
F840 34  |  (XMT BLK)   | PSRS   |     A,X       | SAVE REGISTER CONTENTS
  41 12  |              |        |               |
  42 1F  |              | LDA    |      F        | GET FLAG REGISTER
  43 A8  |              |        |               |
  44 84  |              | ANAI   |               | ENABLE INTERRUPTS
  45 EF  |              |        |      EF       |
  46 1F  |              | LDF    |      A        | STORE FLAG REGISTER
  47 8A  |              |        |               |
  48 8E  |              | LDXI   |               | POINT TO CONTROL STORAGE FOR BLOCK
  49 00  |              |        |     0000      |
  4A 00  |              |        |               |
  4B BD  |              | JS     |      UN       | TRANSMIT A CHARACTER
  4C F8  |              |        |   (XMT CHAR)  |
  4D 60  |              |        |               |
  4E 35  |              | PRLS   |     A,X       | RESTORE REGISER CONTENTS
  4F 12  |              |        |               |
F850 39  |              | RTS    |               |
F851 B6  |  (END XMT?)  | LDAD   |               | GET NUMBER OF CHARACTERS IN LAST BLOCK TRANSMITTED
  52 00  |              |        |     0000      |
  53 00  |              |        |               |
  54 81  |              | CPAI   |               | IS BLOCK STILL BEING TRANSMITTED?
  55 00  |              |        |      00       |
  56 26  |              | JPR    |      NE       | JUMP IF YES TO WAIT FOR END OF TRANSMISSION
  57 F9  |              |        |  (END XMT?)   |
  58 B6  | BUFF EMPTY*  | LDAD   |               | GET ACIA READ STATUS REGISTER
  59 C7  |              |        |     C7FD      |
  5A FD  |              |        |               |
  5B 84  |              | ANAI   |               | MASK TO GET TDRE BIT
  5C 10  |              |        |      10       |
  5D 27  |              | JPR    |      EQ       |
  5E F9  |              |        | BUFF EMPTY*   |
  5F 39  |              | RTS    |               |
F860 34  |  (XMT CHAR)  | PSRS   |     A,X       | SAVE REGISTER CONTENTS
  61 12  |              |        |               |
  62 A6  |              | LDAX   |      0,X      | GET NUMBER OF CHARACTERS IN BLOCK TO BE TRANSMITTED
  63 84  |              |        |               |
  64 81  |              | CPAI   |               | HAS THE LAST CHARACTER BEEN SENT?
  65 00  |              |        |      00       |
  66 27  |              | JPR    |      EQ       | JUMP IF YES
  67 0D  |              |        |    LAST B     |
  68 4A  |              | DCA    |               | DECREMENT COUNT OF BYTES IN BLOCK
  69 A7  |              | STAX   |      ,X+      | STORE BYTE COUNT, POINT TO FIRST CHAR IN BLOCK
  6A 80  |              |        |               |
  6B A6  |              | LDAX   |     (0,X)     | GET NEXT CHAR IN BLOCK
  6C 94  |              |        |               |
  6D 6C  |              | ICMX   |      1,X      | POINT TO NEXT CHAR IN BLOCK
  6E 01  |              |        |               |
  6F B7  |              | STAD   |               | WRITE TO TRANSMIT DATA REGISTER
F870 C7  |              |        |     C7FC      |
  71 FC  |              |        |               |
  72 35  |  RESTORE     | PLRS   |     A,X       | RESTORE REGISTER CONTENTS
  73 12  |              |        |               |
  74 39  |              | RTS    |               |
  75 BD  |  LAST B      | JS     |      UN       | LAST CHAR SENT, DISABLE TRANSMITTER INTERRUPTS
  76 F8  |              |        |   (XMT OFF)   |
  77 95  |              |        |               |
  78 20  |              | JPR    |      UN       | GO RESTORE REGISTER CONTENTS
  79 F8  |              |        |    RESTORE    |
  7A     |              |        |               |
  7B     |              |        |               |
  7C     |              |        |               |
  7D     |              |        |               |
  7E     |              |        |               |
  7F     |              |        |               |
```

Figure 4-13. (Continued)

```
ADRdata  |     LABEL     |INSTR |    MODIFIER    |
---------|---------------|------|----------------|-------------------------------------------------
F880 1F  | (RCV & ECHO)  |LDA   |       F        | GET FLAG REGISTER
  81 A8  |               |      |                |
  82 84  |               |ANAI  |                | ENABLE INTERRUPTS
  83 EF  |               |      |       EF       |
  84 1F  |               |TDF   |       A        | STORE FLAG REGISTER
  85 8A  |               |      |                |
  86 B6  |               |LDAD  |                | GET INITIAL VALUE STORED AT COMMAND REGISTER
  87 F8  |               |      |      F809      |
  88 09  |               |      |                |
  89 8A  |               |ORAI  |                | SET RECEIVER ECHO MODE
  8A 10  |               |      |       10       |
  8B 84  |               |ANAI  |                | SET BITS D3 AND D2 TO 0 AS REQUIRED BY ECHO MODE
  8C F3  |               |      |       F3       |
  8D B7  |               |STAD  |                | STORE AT COMMAND REGISTER
  8E C7  |               |      |      C7FE      |
  8F FE  |               |      |                |
F890 39  |               |RTS   |                |
  91     |               |      |                |
  92     |               |      |                |
  93     |               |      |                |
  94     |               |      |                |
  95 B6  |   (XMT OFF)   |LDAD  |                | GET INITIAL VALUE STORED AT COMMAND REGISTER
  96 F8  |               |      |      F809      |
  97 09  |               |      |                |
  98 84  |               |ANAI  |                | CLEAR TRANSMITTER INTERRUPT CONTROL BITS
  99 F3  |               |      |       F3       |
  9A 8A  |               |ORAI  |                | DISABLE TRANSMITTER INTERRUPT, RTS* LOW
  9B 80  |               |      |       80       |
  9C B7  |               |STAD  |                | STORE AT COMMAND REGISTER
  9D C7  |               |      |      C7FE      |
  9E FE  |               |      |                |
  9F 39  |               |RTS   |                |
F8A0 34  |  (INTERRUPT)  |PSRS  |       A        | SAVE REGISTER CONTENTS (NECESSARY FOR FIRQ
  A1 02  |               |      |                |     INTERRUPT ONLY)
  A2 B6  |               |LDAD  |                | READ ACIA STATUS REGISTER
  A3 C7  |               |      |      C7FD      |
  A4 FD  |               |      |                |
  A5 85  |               |ANAT  |                | IS IRQ BIT SET?
  A6 80  |               |      |       80       |
  A7 27  |               |JPR   |       EQ       | JUMP IF NO (ACIA DIDN'T INTERRUPT)
  A8 08  |               |      |    END INT     |
  A9 85  |               |ANAT  |                | IS RECEIVE DATA REGISTER FULL BIT SET?
  AA 08  |               |      |       08       |
  AB 26  |               |JPR   |       NE       | JUMP IF YES (RECEIVER FULL)
  AC 0C  |               |      |    RCVR FULL   |
  AD 85  |               |ANAT  |                | IS TRANSMITTER DATA REGISTER EMPTY BIT SET?
  AE 10  |               |      |       10       |
  AF 26  |               |JPR   |       NE       | JUMP IF YES (TRANSMITTER EMPTY)
F8B0 03  |               |      |    XMT EMPTY   |
  B1 35  |   END INT     |PLRS  |       A        | RECALL REGISTER CONTENTS
  B2 02  |               |      |                |
  B3 3B  |               |RTI   |                |
  B4 BD  |   XMT EMPTY   |JS    |       UN       | TRANSMIT NEXT CHAR IN BLOCK
  B5 F8  |               |      |    (XMT BLK)   |
  B6 40  |               |      |                |
  B7 20  |               |JPR   |       UN       |
  B8 F8  |               |      |    END INT     |
  B9 B6  |   RCVR FULL   |LDAD  |                | READ CHARACTER FROM RECEIVE DATA REGISTER
  BA C7  |               |      |      C7FC      |
  BB FC  |               |      |                |
  BC 12  |               |NOP   |                | JUMP TO SUBROUTINE "PROCESS CHARACTER"
  BD 12  |               |NOP   |                |    MAY GO HERE
  BE 12  |               |NOP   |                |
  BF 20  |               |JPR   |       UN       |
```

Figure 4-13. (Continued)

ADRdata	LABEL	INSTR	MODIFIER	
F8C0 F0			END INT	
C1				
C2				
C3				
C4				
C5				
C6				
C7				
C8				
C9				
CA				
CB				
CC				
CD				
CE				
CF				
F8D0				
D1				
D2				
D3				
D4				
D5				
D6				
D7				
D8				
D9				
DA				
DB				
DC				
DD				
DE				
DF				
FFE0				
E1				
E2				
E3				
E4				
E5				
E6				
E7				
E8				
E9				
EA				
EB				
EC				
ED				
EE				
EF				
FFF0				
F1				
F2	INTERRUPT VECTOR TABLE			SWI3
F3				
F4				SWI2
F5				
F6				FIRQ
F7				
F8 F8			(INTERRUPT)	IRQ
F9 A0				
FA				SWI
FB				
FC				NMI
FD				
FE				RESTART
FF				

Figure 4-13. (Continued)

Section 5
Maintenance

REFERENCE DRAWINGS

The schematic (figure 5-1) and assembly drawing (figure 5-2) in the following pages are included in this manual FOR REFERENCE USE ONLY. These drawings may differ in some respects from the card you receive from Pro-Log. If you require the specific revision documents of the schematic and assembly for the board shipped, please use the information request form in the back of this manual to order them.

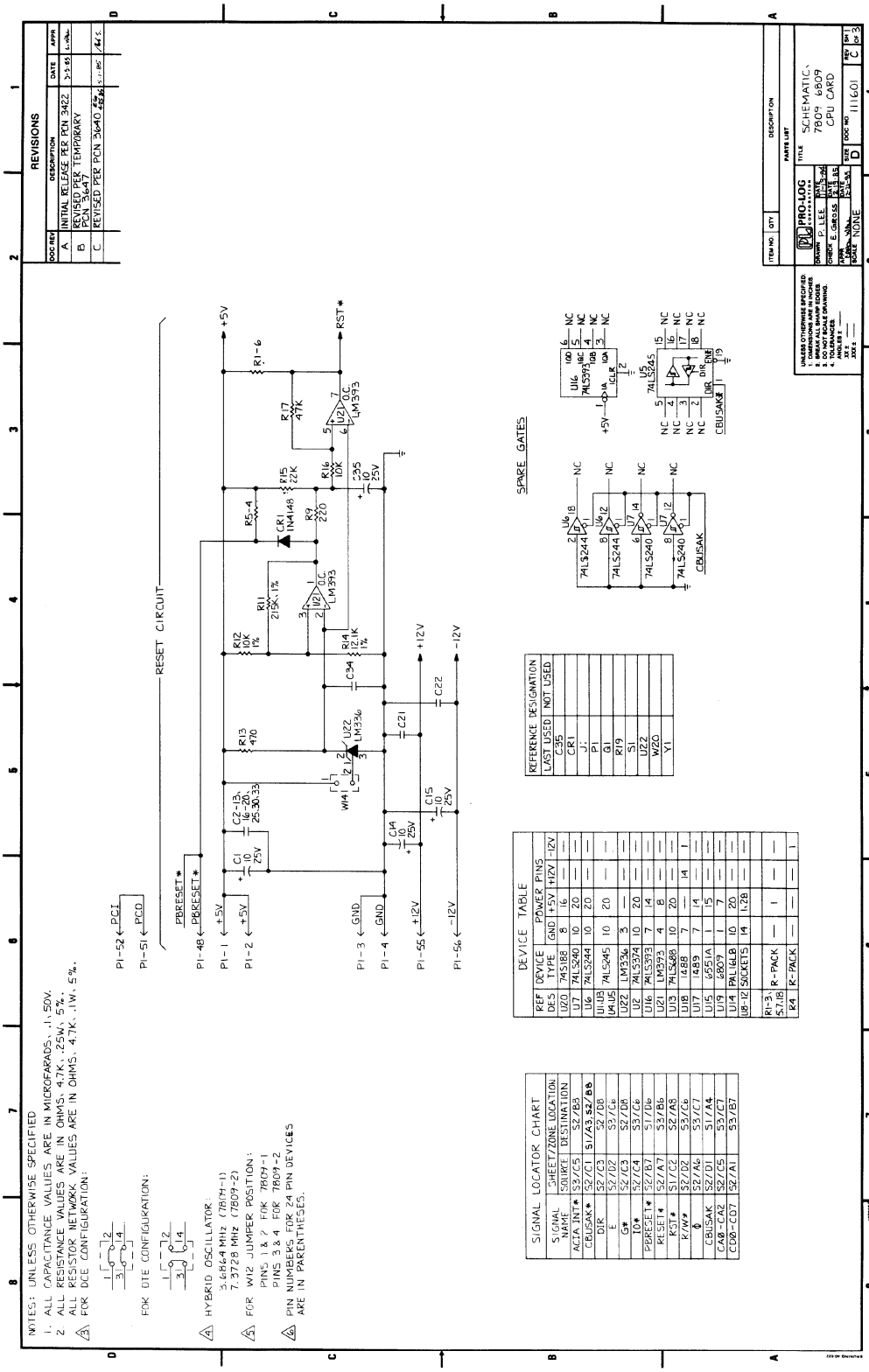


Figure 5-1. 7809 Schematic (Page 1 of 3)

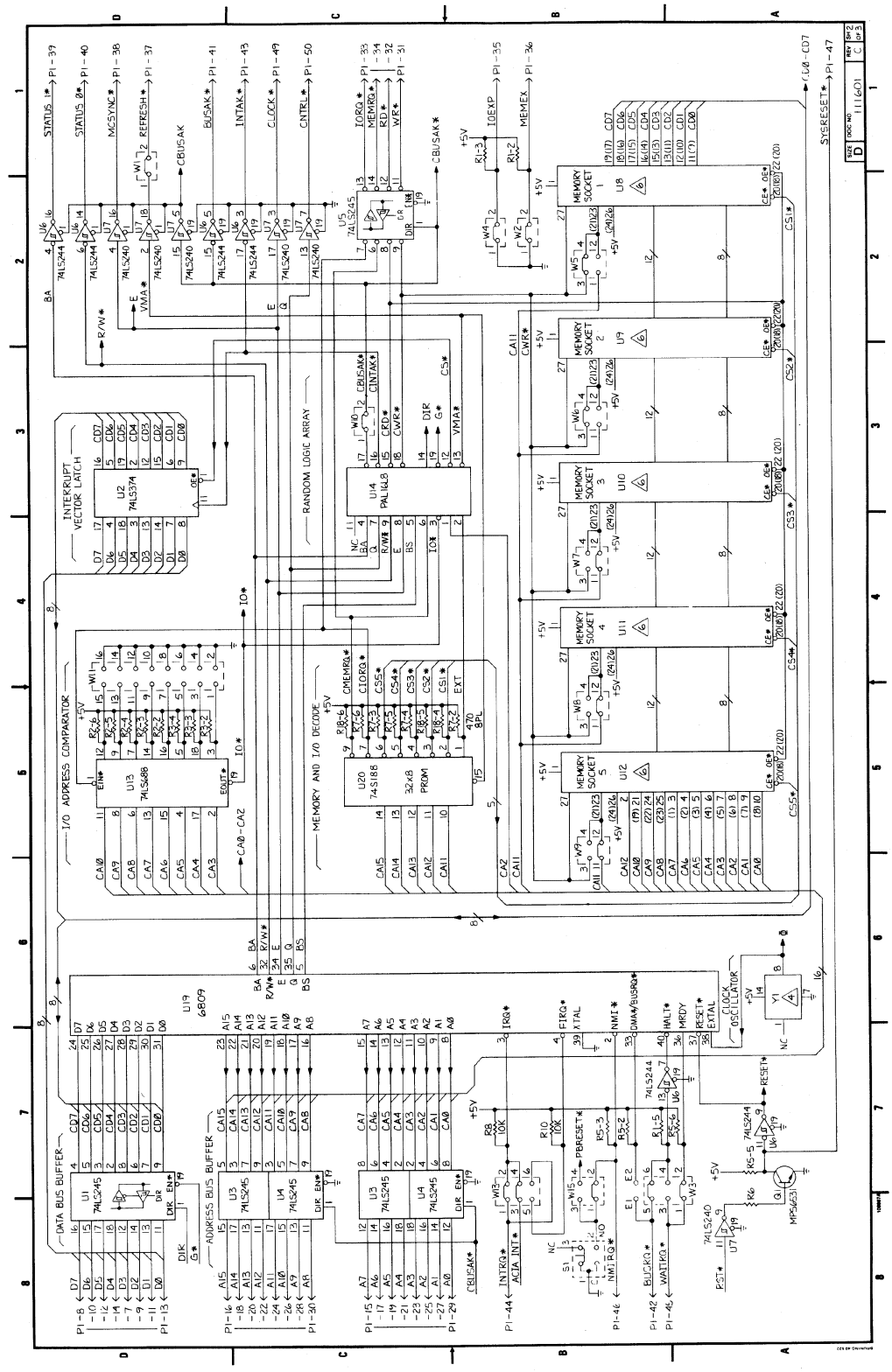


Figure 5-1. 7809 Schematic (Page 2 of 3)

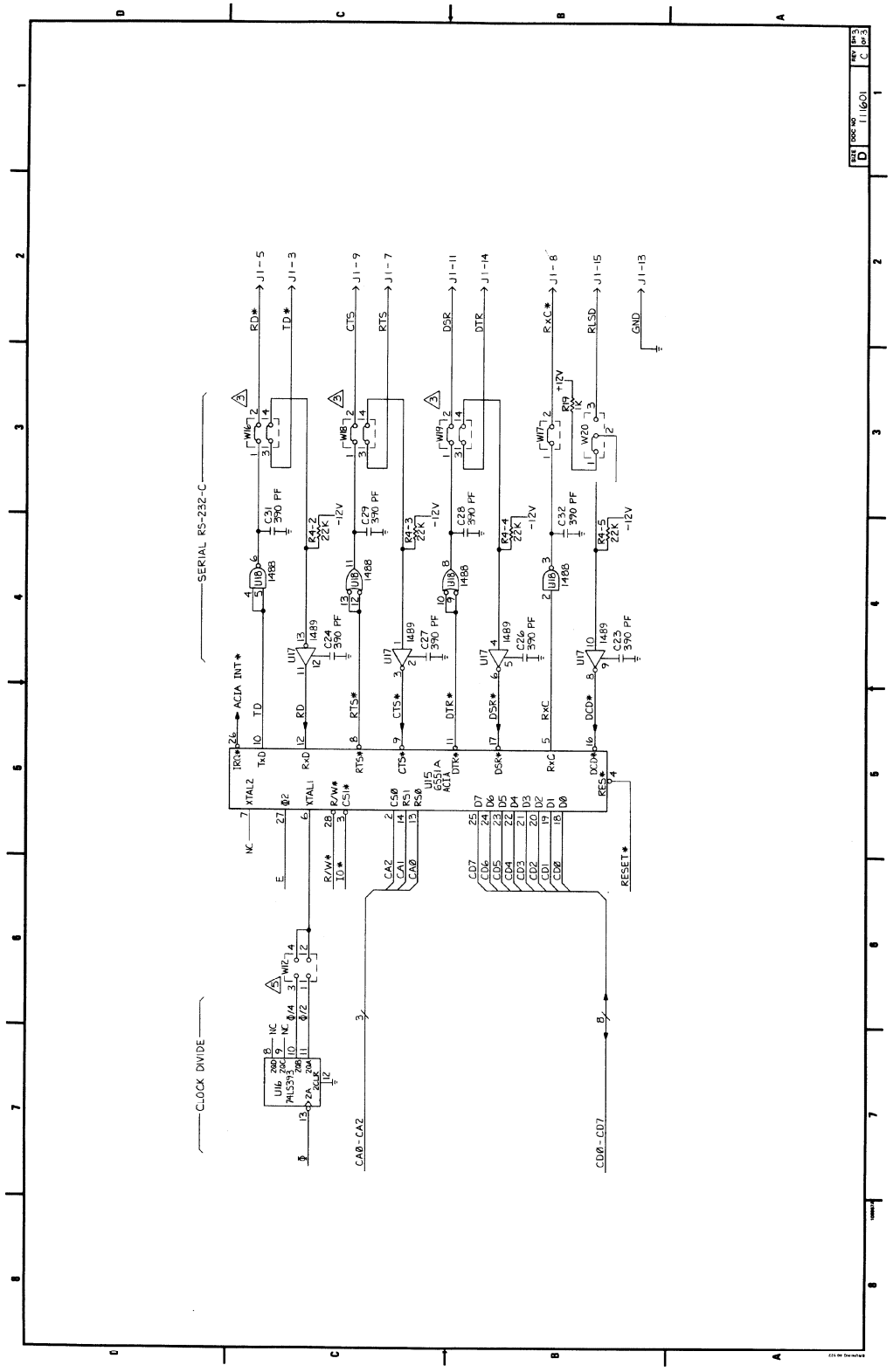
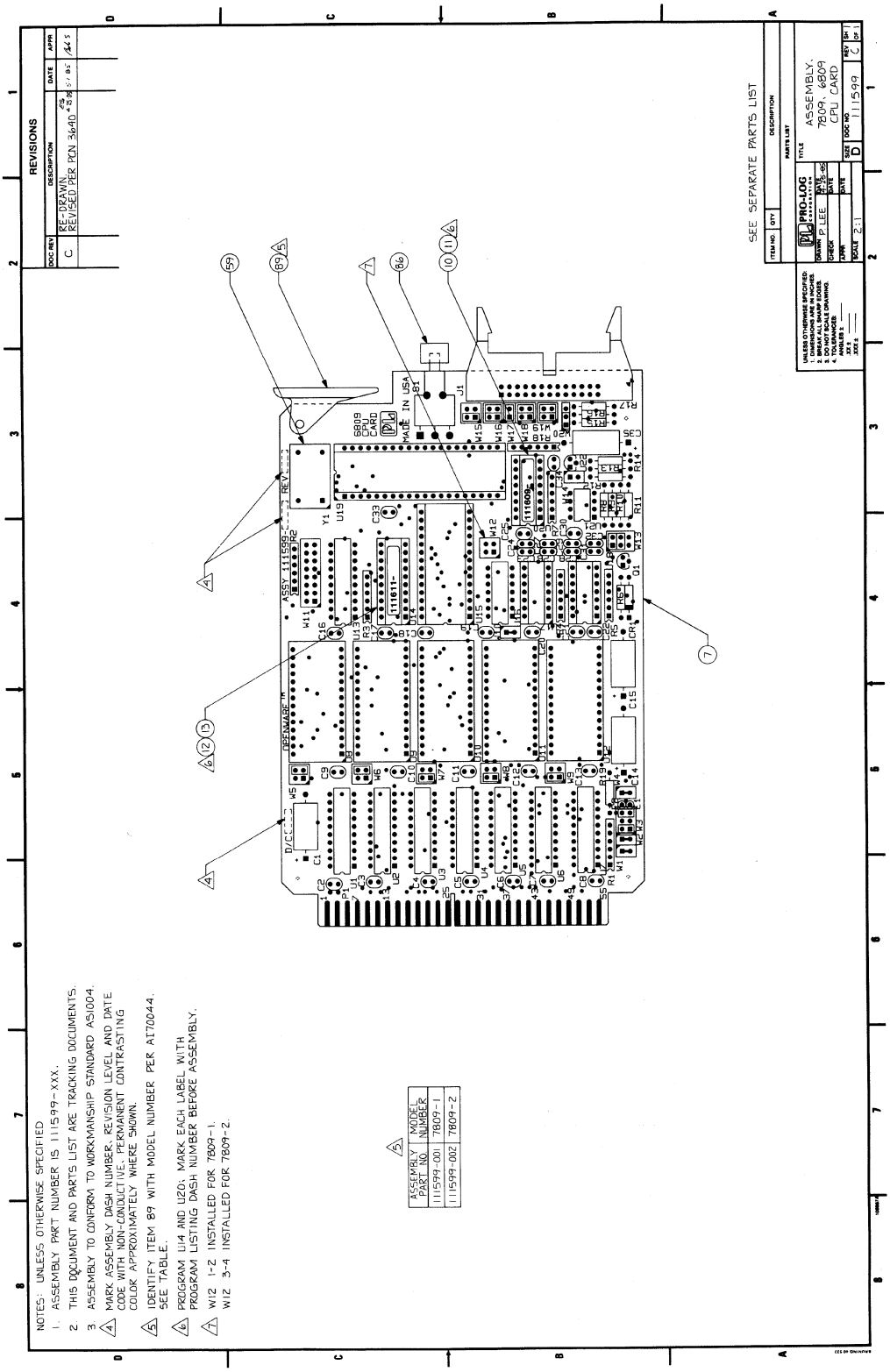


Figure 5-1. 7809 Schematic (Page 3 of 3)


REVISIONS
DOC REV
DESCRIPTION
DATE
APPR
C
RE-DRAWN
REVISED PER PCN 3640
SEE SEPARATE PARTS LIST
PRO-LOG
ASSEMBLY:
7809, 6809
CPU CARD
DOC NO 111599
NOTES: UNLESS OTHERWISE SPECIFIED
1. ASSEMBLY PART NUMBER IS 111599-XXX.
2. THIS DOCUMENT AND PARTS LIST ARE TRACKING DOCUMENTS.
3. ASSEMBLY TO CONFORM TO WORKMANSHIP STANDARD ASI004.
4. MARK ASSEMBLY DASH NUMBER, REVISION LEVEL AND DATE CODE WITH NON-CONDUCTIVE, PERMANENT CONTRASTING COLOR APPROXIMATELY WHERE SHOWN.
5. IDENTIFY ITEM 89 WITH MODEL NUMBER PER AI70044. SEE TABLE.
6. PROGRAM U14 AND U20; MARK EACH LABEL WITH PROGRAM LISTING DASH NUMBER BEFORE ASSEMBLY.
7. W12 1-2 INSTALLED FOR 7809-1.
W12 3-4 INSTALLED FOR 7809-2.
ASSEMBLY PART NO.
MODEL NUMBER
111599-001
7809-1
111599-002
7809-2

FOR REFERENCE ONLY

Figure 5-2. Assembly Drawing (7809)

RETURN FOR REPAIR PROCEDURES

Domestic Customers:

1. Call Customer Support at (800) 538-9570, ext. 3617. In California call (408) 646-3617 from 7 a.m. to 4 p.m. (Pacific Time).
2. Explain the problem and we may be able to solve it on the phone. If not, we will give you a Customer Return Order (CRO) number.

   Mark the CRO number on the shipping label, packing slip, and other paperwork accompanying the return. We cannot accept returns without a CRO.
3. Please be sure to enclose a packing slip with CRO number, serial number of the equipment, if applicable, and reason for return, with the name and telephone number of the person we should contact (preferably the user), if we have any further questions.
4. Pack the equipment in a solid cardboard box secured with packing material.

CAUTION

Loose MOS integrated circuits, or any product containing CMOS integrated circuits, must be protected from electrostatic discharge during handling. Use conductive foam pads or conductive plastic bags, and never place MOS or CMOS circuitry in contact with Styrofoam materials.

5. Ship prepaid and insured to:

   Pro-Log Corporation
   2411 Garden Road
   Monterey, California 93940

   Reference CRO #_____

International Customers:

Equipment repair is handled by your local Pro-Log distributor. If you need to contact Pro-Log, the factory can be reached at any time by TWX at 910-360-7082.

Limited Warranty

Limited Warranty: Seller warrants that the articles furnished hereunder are free from defects in material and workmanship and perform to applicable, published Pro-Log specifications for two years from date of shipment. This warranty is in lieu of any other warranty expressed or implied. In no event will Seller be liable for special or consequential damages as a result of any alleged breach of this warranty provision. The liability of Seller hereunder shall be limited to replacing or repairing, at its option, any defective units which are returned f.o.b. Seller's plant. Equipment or parts which have been subject to abuse, misuse, accident, alteration, neglect, unauthorized repair or installation are not covered by warranty. Seller shall have the right of final determination as to the existence and cause of defect. As to items repaired or replaced, the warranty shall continue in effect for the remainder of the warranty period, or for ninety (90) days following date of shipment by Seller or the repaired or replaced part whichever period is longer. No liability is assumed for expendable items such as lamps and fuses. No warranty is made with respect to custom equipment or products produced to Buyer's specifications except as specifically stated in writing by Seller and contained in the contract.

APPENDIX A
The 6809

THE 6809

This appendix contains general information about the 6809 processor that is not specific to the 7809 card. For further information on the 6809 processor, see References/ Bibliography.

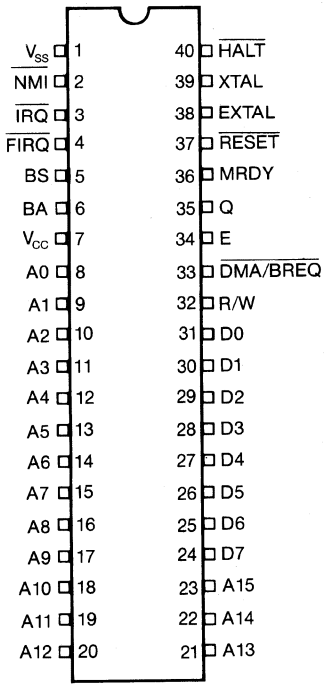


Figure A-1. 6809 Pin Assignments

6809 SIGNAL DESCRIPTIONS

Power (V_{SS}, V_{CC}): V_{SS} supplies ground or 0 volts while V_{CC} is 5V. These two pins supply power to the processor.

Address Bus (A15-A0): These 16 pins put address information from the processor onto the Address Bus. When the processor does not need the bus during the normal address time it does a "dummy access" to memory.

Data Bus (D7-D0): These 8 pins read data from or write data to the system data bus.

Read/Write (R/$\overline{W}$): This pin indicates the direction of travel on the data bus. A low indicates the processor is writing data to the bus. R/$\overline{W}$ is made high impedance (as if not connected) when BA is high (when the bus is not available).

Reset: Resets the processor and fetches the address of the code to begin executing from FFFE and FFFF. When power is first turned on, RESET should be held low until the clock oscillator is fully operational.

Halt: A low at this pin causes the processor to stop running at the end of its current instruction and remain halted without loss of data. The address bus is high impedance (as though not connected). While the halt pin is low the processor doesn't respond to anything except $\overline{DMA}/\overline{BREQ}$. The processor will remember receiving a $\overline{NMI}$ or $\overline{RESET}$ and will act on these later. During $\overline{HALT}$ the processor clocks E and Q continue to be output.

Bus Available, Bus Status: The BA (Bus Available) pin indicates when the processor buses are high impedance (as though not connected). Each time BA changes there will be one clock cycle that is dead while control is transferred.

The BS (Bus Status) pin decoded with BA represents the processor state.

CPU State		CPU State Definition
BA	BS	
0	0	Normal (Running)
0	1	Interrupt or RESET Acknowledge
1	0	SYNC Acknowledge
1	1	HALT or Bus Grant

Interrupt Acknowledge is indicated when the vector address of where to start executing is fetched for $\overline{RESET}$, $\overline{NMI}$, $\overline{FIRQ}$, $\overline{IRQ}$, JI, JI2, JI3.

Sync Acknowledge is indicated while the CPU is waiting for external synchronization on an interrupt line.

Halt/Bus Grant is true when the processor is in a Halt or Bus Grant condition.

Non Maskable Interrupt ($\overline{NMI}$): A negative edge on this pin requests a non maskable interrupt sequence be generated. This interrupt cannot be masked by a program and has a higher priority than $\overline{FIRQ}$, $\overline{IRQ}$, or software interrupts. The entire machine state is stacked on the S stack when the interrupt is recognized.

Fast Interrupt Request ($\overline{FIRQ}$): A low level intiates a fast interrupt if bit F in the flag register is clear. This interrupt has priority over $\overline{IRQ}$ and stacks only the flag register and program counter.

Interrupt Request ($\overline{IRQ}$): A low level initiates an interrupt if bit I in the flag register is clear. Since $\overline{IRQ}$ stacks the entire machine state, it is slower than $\overline{FIRQ}$ and also has a lower priority.

XTAL, EXTAL: These pins connect an on chip oscillator to an external parallel resonant crystal. Or, XTAL can be grounded and a TTL clock can be input at EXTAL. The frequency is four times the bus frequency.

E and Q: These pins provide system clocks. The Q clock leads E by approximately 90°. Addresses are valid with the leading edge of Q. Data is latched on the falling edge of E.

MRDY: When this pin is low it will stretch the quarter cycle when the E clock is high and the Q clock is low. This allows longer data access times. The clocks are stretched in quarter cycle increments, up to a maximum of 10 microseconds.

DMA/BREQ: This pin should go low while Q is high. A low will stop instruction execution at the end of the current cycle. The requesting device has up to 15 bus cycles before the processor retrieves the bus for self-refresh. Each refresh is preceded and followed by a dead cycle and executes one cycle of an instruction. This process of 14 DMA cycles, 1 dead cycle, 1 processor cycle, 1 dead cycle continues. This means the processor is running 17 times slower than normal.

SYSTEM ORGANIZATION

Central Processing Unit

All computers consist of a central processing unit (CPU) and a memory that has a stored sequence of instructions for the CPU. The CPU is operated by a clock circuit to address, fetch, and execute the instructions stored in memory. The CPU fetches an instruction by sending an address from a program address counting register to the program memory. The program memory decodes the address and sends the selected instruction to the CPU. The CPU stores the instruction in an instruction register where it is decoded and executed.

7809 System

The 7809 card is controlled by the 6809 processor chip. The processor performs control and data transfer functions with the logic elements shown in the system data flow diagram, Figure A-2. The processor communicates with memory and I/O ports by connecting appropriate elements of the system to the 8-bit processor bus. Conceptually, the information paths exist as shown in Figure A-2.

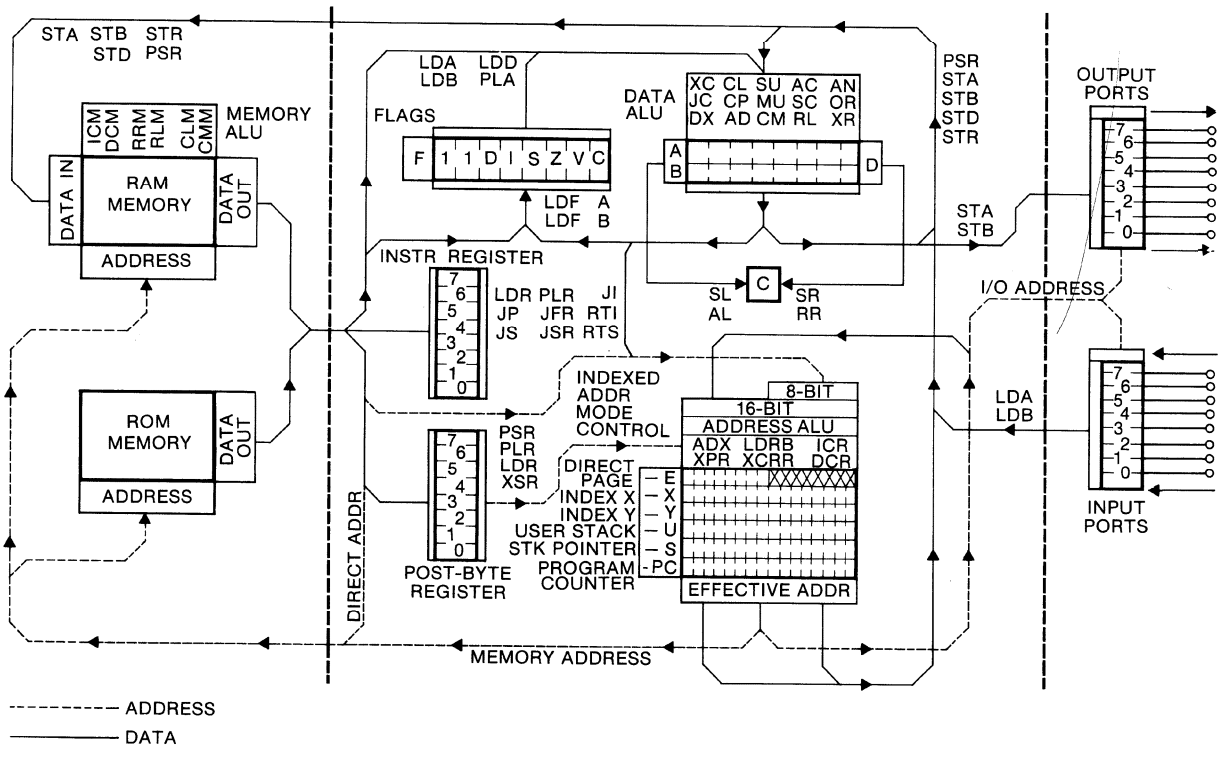


Figure A-2. System Data Flow

6809 Architecture

The 6809 architecture shown consists of an 8-bit instruction register, a 16-bit program address counter, a 16-bit hardware stack pointer register and 16-bit user stack pointer register, two 16-bit index registers, two 8-bit accumulators that can be joined into one 16-bit register, a 16-bit ALU and a 6-bit flag register.

Instruction Register

The 8-bit instruction register provides storage and decoding for the instruction words as they are received from program memory. If the first instruction byte is a 10 or an 11, a second instruction byte follows.

The complete instruction word including addressing may be 40 bits long. Additional words of multibyte instructions bypass the instruction register. These words may be immediate data for registers, line or page and line address for short direct and direct addressing, or a post byte containing register, offset and automatic increment or decrement information for indexing, stack operations and register transfers.

Program Address Counter

The 16-bit program address counter keeps track of the location of the next instruction to be executed from the program memory. The program address counter increments automatically for each instruction word unless the instruction is a JUMP or RETURN which modifies the count by loading a new address. Relative and indexed instructions do not affect the program counter except for JUMPs.

Stack Pointers

Two 16-bit autocounting stack pointers provide the addresses of stack locations in external RAM. The hardware stack pointer(s) is used for controlling subroutines, interrupts and data handling.

Subroutine return addresses are automatically stored on the hardware stack by JUMP to subroutine instructions and retrieved by a RETURN from subroutine.

Interrupt return addresses and the entire machine state (all registers) are automatically stored on the hardware stack by an interrupt or non-maskable interrupt. A fast interrupt stores only a return address and the flag register. These stored registers are automatically retrieved by a RETURN from interrupt.

Any register can be stored on the hardware stack or the user stack which is pointed to by the user stack pointer U. Registers are stored on either stack with a PUSH instruction and retrieved by a PULL instruction.

Direct Page Register

An 8-bit register is loaded with an 8-bit base address. Then, using short direct addressing, this 8-bit register is concatenated with the byte immediately following an opcode. This allows accessing 256 locations within any one of 256 pages and two-byte instructions.

Index Registers

In indexed addressing modes, one of the 16-bit pointer registers (X, Y, U, or S) or even the program counter (PC) is used to calculate the effective address of an instruction operand. In constant offset indexing, no offset, 5-bit, 8-bit, or 16-bit offsets are added to the index register temporarily to form a memory address. Indexing also includes accumulator offset from the index register, no offset with automatic incrementing or decrementing of the index register and indirection which can be used with all indexed addressing modes.

Data ALU and Accumulators

The two accumulators and their associated carry bit are part of the data ALU. The data ALU provides add and subtract with or without carry, AND, OR, exclusive OR, compare, complement, negate, test, bit test, clear, increment, decrement, and multiply. The arithmetic and logic operations can be performed on the accumulators from memory either indexed, short and long direct or immediately from the second word of the instruction. The accumulators can also be shifted and rotated to the left or to the right either with or without carry. The A accumulator can also be decimal adjusted.

Flag Register (Status)

The flag register contains six status flags; Carry (C), Zero (Z), Sign (S), Overflow (V), Decimal (D) and Interrupt (I). The C, Z, S, and V flags can be tested by the relative jump instructions to make program decisions. The D flag affects the operation of the Decimal Adjust and the I flag enables the maskable interrupts.

Memory

The 16-bit addressing capability allows 64K bytes of memory which can be any combination of ROM or RAM. RAM is required for stack operations to allow the use of subroutines and interrupts. Memory is shared by data, programs and I/O. The instruction set allows immediate, relative, indexed, and long and short direct addressing. The relative and indexed memory modes allow "relocatable programs" and "dynamic memory allocation" techniques to be implemented. The short form of addressing, addresses within the page contained in the direct page register. One page can be used as a workspace when efficient programs are desired. Indexed and long and short addressing can also be used in indirect addressing.

Inputs and Outputs

I/O is memory mapped i.e., there are no separate I/O instructions. I/O ports use the same data, address, and control lines as memory. All memory instructions can be used for manipulating the I/O ports.

Interrupt

The 6809 has three wired interrupt inputs, each of which causes the program to retrieve a fixed address from Page FF. One of these wired interrupts is non-maskable.

The entire machine state (all registers and program return address) is automatically pushed onto the stack by two of the interrupt instructions and one of the HALT instructions. The fast interrupt stores only a return address and the flag register.

Summary

The 6809 has an architecture that favors data processing applications. The architecture has short and long relative addressing and indexed addressing modes for "relocatable programs" and "dynamic memory allocation". The architecture is memory oriented with only two 8-bit accumulators and the remainder of the registers 16 bit. However, the register-to-register instructions and instructions that treat the two accumulators as a single register makes the processor look register oriented. The powerful instruction set and indexing make this a good general purpose architecture. Efficiency suffers from some 16 bit instructions and an external subroutine stack but is aided by short addressing and automatic incrementing and decrementing of pointers.

PROGRAMMING METHODS

Position Independence

A program is said to be "position-independent" if it will run correctly when the same machine code is positioned arbitrarily in memory. Such a program is useful in many different hardware configurations, and might be copied from a disk into RAM when the operating system first sees a request to use a system utility. Position-independent programs never use absolute (direct or short direct) addressing; instead, they use inherent, immediate, register, indexed and relative modes. In particular, there should be no jump (absolute) or jump to subroutine instructions, nor should absolute addresses be used. A position-independent program is almost always preferable to a position-dependent program (although position-independent code is usually 5 to 10% slower than normal code).

Modular Programming

Modular programming is another indication of quality code. A module is a program element which can be easily disconnected from the rest of the program either for re-use in a new environment or for replacement. A module is usually a subroutine (although a subroutine is not necessarily a module). Frequently, the programmer isolates register changes internal to the module by pushing these registers onto the stack upon entry, and pulling them off the stack before the return. Isolating register changes in the called module, to that module alone, allows the code in the calling program to be more easily analyzed since it can be assumed that all registers (except those specifically used for parameter transfer) are unchanged by each called module. This leaves the processor's registers free at each level for loop counts, address comparisons, etc.

Local Storage: A clean method for allocating "local" storage is required both by position-independent programs as well as modular programs. Local or temporary storage is used to hold values only during execution of a module (or called modules) and is released upon return. One way to allocate local storage is to decrement the hardware stack pointer(s) by the number of bytes needed. Interrupts will then leave this area intact and it can be de-allocated on exiting the module. A module will almost always need more temporary storage than just the CPU registers.

Global Storage: Even in a modular environment there may be a need for "global" values which are accessible by many modules within a given system. These provide a convenient means for storing values from one invocation to another invocation of the same routine. Global storage may be created as local storage at some level, and a pointer register (usually U) used to point at this area. This register is passed unchanged in all subroutines, and may be used to index into the global area.

Re-entrancy/Recursion

Many programs will eventually involve execution in an interrupt-driven environment. If the interrupt handlers are complex, they might well call the same routine which has just been interrupted. Therefore, to protect present programs against certain obsolescence, all progams should be written to be re-entrant. A re-entrant routine allocates different local variable storage upon each entry. Thus, a later entry does not destroy the processing associated with an earlier entry.

The same technique which was implemented to allow re-entrancy also allows recursion. A recursive routine is defined as a routine that calls itself.

6809 Capabilities

The following paragraphs briefly explain how the 6809 is used with the programming techniques mentioned earlier.

Module Construction: A module can be defined as a logically self-contained and discrete part of a larger program. The use of parameters, local storage, and global storage by a program module is given in the following paragraphs. Since registers will be used inside the module (essentially a form of local storage), the first thing that is usually done at entry to a module is to push (save) them on to the stack. This can be done with one instruction (e.g., PSRS Y,X,B,A). After the body of the module is executed, the saved registers are collected, and a subroutine return is performed, at one time, by pulling the program counter from the stack (e.g., PLRS A,B,X,Y,PC).

Parameters: Parameters may be passed to or from modules either in registers, if they will provide sufficient storage for parameter passage, or on the stack. If parameters are passed on the stack, they are placed there before calling the lower level module. The called module is then written to use local storage inside the stack as needed (e.g., ADAX offset,S). Notice that the required offset consists of the number of bytes pushed (upon entry), plus two from the stacked return address, plus the data offset at the time of the call. This value may be calculated, by hand, by drawing a "stack picture" diagram representing module entry, and assigning convenient mnemonics to these offsets with the assembler. Returned parameters replace those sent to the routine. If more parameters are to be returned on the stack than would normally be sent, space for their return is allocated by the calling routine before the actual call (if four additional bytes are to be returned, the caller would execute LDSEX -4,S to acquire the additional storage).

Local Storage: Local storage is acquired from the stack while the present routine is executing and then returned to the stack prior to exit. The act of pushing registers which will be used in later calculations essentially saves those registers in temporary local storage. Additional local storage can easily be acquired from the stack e.g., executing LDSEX -2048,S acquires a buffer area running from 0,S to 2047,S inclusive. Any byte in this area may be accessed directly by any instruction which has an indexed addressing mode. At the end of the routine, the area acquired for local storage is released (e.g. LDSEX 2048,S) prior to the final pull. For cleaner programs, local storage should be allocated at entry to the module and released at the exit of the module.

Global Storage: The area required for global storage is also most effectively acquired from the stack, probably by the highest level routine in the standard package. Although this is local storage to the highest level routine, it becomes "global" by positioning a register to point at this storage (sometimes referred to as a stack mark), then establishing the convention that all modules pass that same pointer value when calling lower level modules. In practice, it is convenient to leave this stack mark register unchanged in all modules, especially if global accesses are common. The highest level routine in the standard package would execute the following sequence upon entry (to initialize the global area):

```
PSRS   U     Higher level mark, if any
LDU    S     New stack mark
LDSEX  -17,U Allocate global storage
```

Note that the U register now defines 17 bytes of locally allocated (permanent) globals (which are -1,U through -17,U) as well as other external globals (2,U and above) which have been passed on the stack by the routine which called the standard package. Any global may be accessed by any module using exactly the same offset value at any level. Furthermore, the values stacked prior to invoking the standard package may include pointers to data or I/O peripherals. Any indexed operation may be performed indexed indirect through those pointers, which means, for example, that the module need know nothing about the actual hardware configuration, except that (upon entry) the pointer to an I/O register has been placed at a given location on the stack.

Position-Independent Code

Position-independent code means that the same language code can be placed anywhere in memory and still function correctly. The 6809 has a long relative (16-bit offset) jump mode along with a short relative (8-bit offset) mode, plus program-counter relative addressing. Program-counter relative addressing uses the program counter like an indexable register, which allows all instructions that reference memory to also reference data relative to the program counter. The 6809 also has load register effective (LDrE) instructions which allow the user to point to data in a ROM in a position-independent manner.

An important rule for generating position-independent code is: **NEVER USE ABSOLUTE ADDRESSING.**

Program-counter relative addressing on the 6809 is a form of indexed addressing that uses the program counter as the base register for a constant-offset indexing operation. The PCR indexed addressing form can be used to point at any location relative to the program regardless of position in memory. The PCR form of indexed addressing allows access to tables within the program space in a position-independent manner via use of the load register effective instruction.

In a program which is completely position-independent, some absolute locations are usually required, particularly for I/O. If the locations of I/O devices are placed on the stack (as globals) by a small setup routine before the standard package is invoked, all internal modules can do their I/O through that pointer (e.g., STA (ACIAD,U), allowing the hardware to be easily changed, if desired. Only the single, small and obvious setup routine need be rewritten for each different hardware configuration.

Global, permanent, and temporary values need to be easily available in a position-independent manner. Use the stack for this data since the stacked data is directly accessible. Stack the absolute address of I/O devices before calling any standard software package since the package can use the stacked addresses for I/O in any system.

The LDrE instructions allow access to tables, data, or immediate values in the text of the program in a position-independent manner as shown in the following example:

```
                    •
                    •
                    •
        LDXEX               MSG1,PCR
        JSRL        UN      PDATA
                    •
                    •
                    •
MSG1    ASCII TABLE         /PRINT THIS/
```

Here we wish to point at a message to be printed from the body of the program. By writing "MSG1,PCR", we compute the distance between the present address (the address of the JSRL) and MSG1. This result is inserted as a constant into the LDrE instruction which will be indexed from the program counter value at the time of execution. Now, no matter where the code is located, when it is executed the computer offset from the program counter will point at MSG1. This code is position-independent.

It is common to use space in the hardware stack for temporary storage. Space is made for temporary variables from 0,S through TEMP -1,S by decrementing the stack pointer equal to the length of required storage. We could use:

```
LDSEX   -TEMP,S.
```

Not only does this facilitate position-independent code but it is structured and helps re-entrancy and recursion.

Re-entrant Programs

A program that can be executed by several different users sharing the same copy of it in memory is called re-entrant. This is important for interrupt driven systems. This method saves considerable memory space, especially with large interrupt routines. Stacks are required for re-entrant programs, and the 6809 can support up to four stacks by using X and Y index registers as stack pointers.

Stacks are simple and convenient mechanisms for generating re-entrant programs. Subroutines which use stacks for passing parameters and results can be easily made to be re-entrant. Stack accesses use the indexed addressing mode for fast, efficient execution. Stack addressing is quick.

Pure code, or code that is not self-modifying, is mandatory to produce re-entrant code. No internal information within the code is subject to modification. Re-entrant code never has internal temporary storage, is simpler to debug, can be placed in ROM, and must be interruptable.

Recursive Programs

A recursive program is one that can call itself. They are quite convenient for certain arithmetic functions such as computing factorials. As with re-entrant programming, stacks are very useful for this technique.

Loops

The usual structured loops (i.e., REPEAT . . . UNTIL, WHILE . . . DO, FOR . . . , etc.) are available in assembly language in exactly the same way a high-level language compiler could translate the construct for execution on the target machine. Using a FOR . . . NEXT loop as an example, it is possible to push the loop count, increment value, and termination value on the stack as variables local to that loop. On each pass through the loop, the working register is saved, the loop count picked up, the increment added in, and the result compared to the termination value. Based on this comparison, the loop counter might be updated, the working register recovered and the loop resumed, or the working register recovered and the loop variables deallocated.

Stack Programming

Many microprocessor applications require data stored as continous pieces of information in memory. The data may be temporary, that is, subject to change or it may be permanent. Temporary data will most likely be stored in RAM. Permanent data will most likely be stored in ROM.

It is important to allow the main program as well as subroutines access to this block of data, especially if arguments are to be passed from the main program to the subroutines and vice versa.

6809 Stacking Operations: Stack pointers are markers which point to the stack and its internal contents. Although all four index registers may be used as stack registers, the S (hardware stack pointer) and the U (user stack pointer) registers are generally preferred because the push and pull instructions apply to these registers. Both are 16-bit indexable registers. The processor uses the S register automatically during interrupts and subroutine calls. The U register is free for any purpose needed. It is not affected by interrupts or subroutine calls implemented by the hardware.

Either stack pointer can be specified as the base address in indexed addressing. One use of the indirect addressing mode uses stack pointers to allow addresses of data to be passed to a subroutine on a stack as arguments to a subroutine. The subroutine can now reference the data with one instruction. High-level language calls that pass arguments by reference are now more efficiently coded. Also, each stack push or pull operation in a program uses a postbyte which specifies any register or set of registers to be pushed or pulled from either

stack. With this option, the overhead associated with subroutine calls in both assembly and high-level language programs is greatly decreased. In fact, with the large number of instructions that use autoincrement and autodecrement, the 6809 can emulate a true stack computer architecture.

Using the S or U stack pointer, the order in which the registers are pushed or pulled is shown in Figure A-3. Notice that we push "onto" the stack towards decreasing memory locations. The program counter is pushed first. Then the stack pointer is decremented and the "other" stack pointer is pushed onto the stack. Decrementing and storing continues until all the registers requested by the postbyte are pushed onto the stack. The stack pointer points to the top of the stack after the push operation.

The stacking order is specified by the processor and is identical to the order used for all hardware and software interrupts. The same order is used even if a subset of the registers is pushed.

Good programming practice dictates the use of the hardware stack for temporary storage. To reserve space, decrement the stack pointer by the amount of storage required with the instruction LDSEX -TEMPS,S. This instruction makes space for temporary variables from 0,S through TEMPS -1,S.

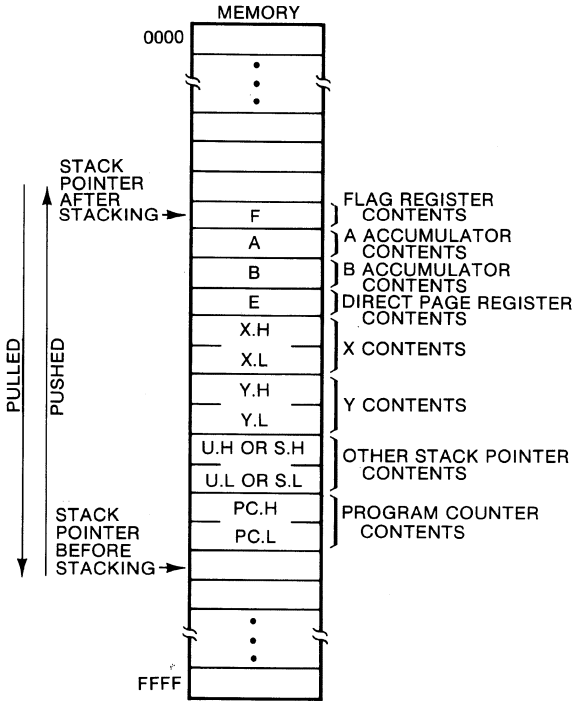


Figure A-3. Stacking Order

ADDRESSING MODES

The 6809 has 59 basic instructions; however, it recognizes 1464 different variations of instructions and addressing modes. The addressing modes support modern programming techniques. The following addressing modes are available on the 6809:

- Inherent (Includes Accumulator)
- Immediate
- Direct
  - Direct Indirect
- Short Direct
- Register
- Indexed
  - Zero-Offset
  - Constant Offset
  - Accumulator Offset
  - Auto Increment/Decrement
  - Indexed Indirect
- Relative
  - Short/Long Relative Jumps
  - Program Counter Relative Addressing

Inherent (Includes Accumulator)

In this addressing mode, the opcode of the instruction contains all the address information necessary. Examples of Inherent Addressing are:

ADX B, AJAD, JI, SRAA, and CLB.

Immediate Addressing

In Immediate Addressing, the effective address of the data is the location immediately following the opcode (i.e., the data to be used in the instruction immediately follows the opcode of the instruction). The 6809 uses both 8 and 16-bit immediate values depending on the size of argument specified by the opcode. Examples of instructions with Immediate Addressing are:

86 20	LDAI	20
8E F0 00	LDXI	F000
10 8E— —	LDYI	CAT

LDAI 20 is load A immediate with 20. The immediate value of hexadecimal 20 is loaded into register A.

Direct Addressing

In Direct Addressing, the contents of the two bytes immediately following the opcode fully specify the 16-bit effective address used by the instruction. Note that the address generated by a direct instruction defines an absolute address and is not position dependent. Examples of Direct Addressing include:

FC 20 00	LDDD	2000	or	LDD	2000
B6 —	LDAD	CAT	or	LDA	CAT
BF — —	STXD	MOUSE	or	STX	MOUSE

The D indicator for this mode can be omitted because this is the default mode. LDDD 2000 is load D direct from 2000. This loads register D with the data at address 2000.

Direct Indirect

As a special case of indexed addressing (discussed below), "1" level of indirection may be added to Direct Addressing. In Direct Indirect, the two bytes following the post byte of an Indexed instruction contain the address of the data.

```
AE 9F FF EE   LDXX    (FFEE)
A6 9F —       LDAX    (CAT)
EF 9F — —     STUX    (DOG)
```

LDXD (FFEE) is load X direct from FFEE indirectly. (The directly following address is treated indirectly.) This instruction loads X with the data contained at the address contained in location FFEE.

Short Direct Addressing

Short Direct addressing is similar to Direct addressing except that only one byte of address follows the opcode. This byte specifies the lower 8-bits of the address to be used since only one byte of address is required in short direct addressing. The upper 8-bits of the address are supplied by the direct page register (E register). Since only one byte of address is required in short direct addressing, this mode requires less memory and executes faster than direct addressing. Of course, only 256 locations (one page) can be accessed without redefining the contents of the direct page register. Since the direct page register is set to 00 on Reset, short direct addressing on the 6809 is compatible with direct addressing on the 6800.

Indirection is not allowed in short direct addressing. Some examples of short direct addressing are:

```
DC 20 LDDS  20
96 30 LDAS  30
```

LDDS is load D short direct from 20. This takes the contents of the direct page register as the upper byte of address and 20 as the lower byte of the address containing the data to load D with.

Register Addressing

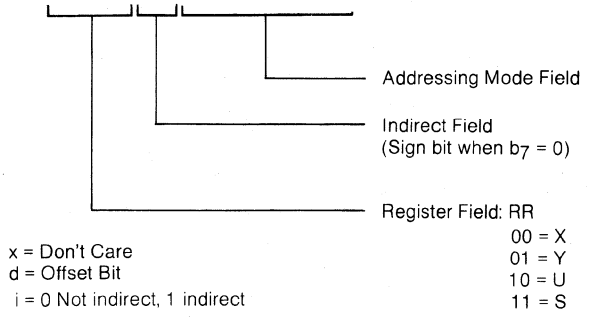
Some opcodes are followed by a byte that defines a register or set of registers to be used by the instruction. This is called a post byte. Some examples of register addressing are:

```
1F  21          LDX  Y    Load register X with the
                          contents of register Y.
1E  89          XCA  B    Exchange the contents of
                          register A and B (or 1E 98)
34  36  PSrS   A,B,X,Y   Push Y,X,B, and A onto S
37  36  PLrU   X,Y,D     Pull D,X, and Y from U
```

Indexed Addressing

In all indexed addressing, one of the pointer registers (X,Y,U,S, and sometimes PC) is used in a calculation of the effective address of the operand to be used by the instruction. Five basic types of indexing are available and are discussed below. The post byte of an indexed instruction specifies the basic type and variation of the addressing mode as well as the pointer register to be used. Figure A-4 lists the legal formats for the post byte. Figure A-5 gives the assembler form and the number of cycles and bytes added to the basic values for indexed addressing for each variation.

Post Byte Register BIT								Indexed Addressing Mode
7	6	5	4	3	2	1	0	
0	R	R	d	d	d	d	d	EA = R + 5 Bit Offset
1	R	R	0	0	0	0	0	,R+
1	R	R	i	0	0	0	1	,R++
1	R	R	0	0	0	1	0	,—R
1	R	R	i	0	0	1	1	,——R
1	R	R	i	0	1	0	0	EA = ,R + 0 Offset
1	R	R	i	0	1	0	1	EA = ,R + ACCB Offset
1	R	R	i	0	1	1	0	EA = ,R + ACCA Offset
1	R	R	i	1	0	0	0	EA = ,R + 8 Bit Offset
1	R	R	i	1	0	0	1	EA = ,R + 16 Bit Offset
1	R	R	i	1	0	1	1	EA = ,R + D Offset
1	X	X	i	1	1	0	0	EA = ,PC + 8 Bit Offset
1	X	X	i	1	1	0	1	EA = ,PC + 16 Bit Offset
1	R	R	i	1	1	1	1	EA = ,(Address)



x = Don't Care
d = Offset Bit
i = 0 Not indirect, 1 indirect

Figure A-4. Indexed Addressing Post Byte Register Bit Assignments

Type	Forms	Non Indirect				Indirect			
		Assembler Form	Postbyte OP Code	+ —	+ /	Assembler Form	Postbyte OP Code	+ —	+ /
Constant Offset From R (2's Complement Offsets)	No Offset	,R	1RR00100	0	0	(,R)	1RR10100	3	0
	5 Bit Offset	n,R	0RRnnnnn	1	0	defaults to 8-bit			
	8 Bit Offset	n, R	1RR01000	1	1	(n, R)	1RR11000	4	1
	16 Bit Offset	n, R	1RR01001	4	2	(n, R)	1RR11001	7	2
Accumulator Offset from R (2's Complement Offsets)	A Register Offset	A, R	1RR00110	1	0	(A, R)	1RR10110	4	0
	B Register Offset	B, R	1RR00101	1	0	(B, R)	1RR10101	4	0
	D Register Offset	D, R	1RR01011	4	0	(D, R)	1RR11011	7	0
Auto Increment/ Decrement R	Increment By 1	,R+	1RR00000	2	0	not allowed			
	Increment By 2	,R++	1RR00001	3	0	(,R++)	1RR10001	6	0
	Decrement By 1	,-R	1RR00010	2	0	not allowed			
	Decrement By 2	,--R	1RR00011	3	0	(,--R)	1RR10011	6	0
Constant Offsets From PC (2's Complement Offsets)	8 Bit Offset	n, PCR	1xx01100	1	1	(n, PCR)	1xx11100	4	1
	16 Bit Offset	n, PCR	1xx01101	5	2	(n, PCR)	1xx11101	8	2
Extended Indirect	16 Bit Address	—	—	—	—	(n)	10011111	5	2

R = X, Y, U or S
x = Don't Care

RR:
00=X
01=Y
10=U
11=S

$\pm$ and $\frac{+}{}$ indicate the number of additional cycles and bytes for the particular variation.

Figure A-5. Indexed Addressing Mode

Zero-Offset Indexed

In this mode, the selected pointer register contains the effective address of the data to be used by the instruction. This is the fastest indexing mode. Examples are:

```
AE  84  LDDX  0,X
A6  E4  LDAX  S
```

LDDX 0,X is load D indexed with 0,X. The contents of X plus 0 is the address of the data to load D with.

Constant Offset Indexed

In this mode, a two's-complement offset and the contents of one of the pointer registers are added to form the effective address of the operand. The pointer register's initial content is unchanged by the addition.

Three sizes of offsets are available:

5-bit (-16 to +15)
8-bit (-128 to +127)
16-bit (-32768 to +32767)

The two's complement 5-bit offset is included in the post byte and, therefore, is most efficient in use of bytes and cycles. The two's complement 8-bit offset is contained in a single byte following the post byte. The two's complement 16-bit offset is in the two bytes following the post byte. In most cases the programmer need not be concerned with the size of this offset since the assembler will select the optimal size automatically.

Examples of constant-offset indexing are:

```
A6 88 23        LDAX   23,X
AE 7E           LDXX   -2,S
10 AE 89 03 00  LDYX   300,X
EE A8 —         LDUX   CAT,Y
```

LDAX 23,X is load A indexed with 23,X. The contents of X are added to 23 to get the address of the data to be loaded into A. Register X is not changed by this instruction.

Accumulator-Offset Indexed

This mode is similar to constant offset indexed except that the two's-complement value in one of the accumulators (A, B or D) and the contents of one of the pointer registers are added to form the effective address of the operand. The contents of both the accumulator and the pointer register are unchanged by the addition. The post byte specifies which accumulator to use as an offset and no additional bytes are required. The advantage of an accumulator offset is that the value of the offset can be calculated by a program at run-time. Some examples are:

```
A6 A5    LDAX   B,Y
AE AB    LDXX   D,Y
30 85    LDXEX  B,X
```

LDAX B,Y is load A indexed with B,Y. The contents of register B are temporarily added to the contents of register Y to form the address of the data to be loaded into register A. The contents of registers B and Y are unchanged by this operation.

LDXEX B,X is load X with the effective address B,X. The contents of B are temporarily added to the contents of register X to form an address, this effective address is loaded into X.

Auto Increment/Decrement Indexed

In the auto increment addressing mode, the pointer register contains the address of the operand. Then, after the pointer register is used, it is incremented by one or two. This addressing mode is useful in stepping through tables, moving data, or for the creation of software stacks. In auto decrement, the pointer register is decremented prior to use as the address of the data. The use of auto decrement is similar to that of auto increment; but the tables, etc., are scanned from the "High" to "Low" addresses. The size of the increment/decrement can be either one or two to allow for tables of either 8 or 16-bit data to be accessed and is selectable by the programmer. The pre-decrement, post-increment nature of these modes allow them to be used to create additional software stacks that behave identically to the U and S stacks. Some examples of the auto increment/decrement addressing modes are:

```
A6 80       LDAX   ,X+
ED A1       STDX   ,Y++
E6 A2       LDBX   ,-Y
AE E3       LDXX   ,--S
```

Indexed Indirect

All of the indexing modes with the exception of auto increment/decrement by one, or a ± 4-bit offset, may have an additional level of indirection specified. In indirect addressing, the effective address is contained at the location specified by the contents of the Index register plus any offset. In the example below, the A accumulator is loaded indirectly using an effective address calculated from the Index register and an offset.

```
   Before execution
   A = XX (don't care)
   X = F000

0100 LDAX (10,X)          EA is now F010

F010 F1                   F150 is now the
F011 50                   new EA

F150    AA
        After Execution
        A = AA Actual Data Loaded
        X = F000
```

All modes of indexed indirect are included except those which are meaningless (e.g., auto increment/decrement by 1 indirect). Some examples of indexed indirect are:

```
A6 94       LDAX   (,X)
EC F8 10    LDDX   (10,S)
A6 B5       LDAX   (B,Y)
EC 91       LDDX   (,X++)
```

Relative Addressing

The byte(s) following the jump relative opcode is (are) treated as a signed offset which may be added to the program counter. If the jump condition is true, then the calculated address (PC + signed offset) is loaded into the program counter. Program execution continues at the new location as indicated by the PC; short (1 byte offset) and long (2 bytes offset) relative addressing modes are available. All of memory can be reached in long relative addressing as an effective address is interpreted modulo 2^{16}. Some examples of relative addressing are:

```
27 —                  JPR EQ    CAT      (short)
2E —                  JPR GT    DOG      (short)
10 27 — —   CAT       JPRL EQ   RAT      (long)
10 2E — —   DOG       JPRL GT   RABBIT   (long)
                      •
                      •
                      •
12          RAT       NOP
12          RABBIT    NOP
```

Program Counter Relative

The PC can be used as the pointer register with 8 or 16-bit signed offsets. As in relative addressing, the offset is added to the current PC to create the effective address. The effective address is then used as the address of the operand or data. Program Counter Relative Addressing is used for writing position independent programs. Tables related to a particular routine will maintain the same relationship after the routine is moved, if referenced relative to the Program Counter. Examples are:

```
A6 8C —     LDAX     CAT,PC
308C —      LDXEX    TABLE,PC
```

LDAX CAT,PC is load A indexed with CAT,PC. This instruction temporarily adds the value CAT to the program counter to get an address. A is loaded with the data contained at the address.

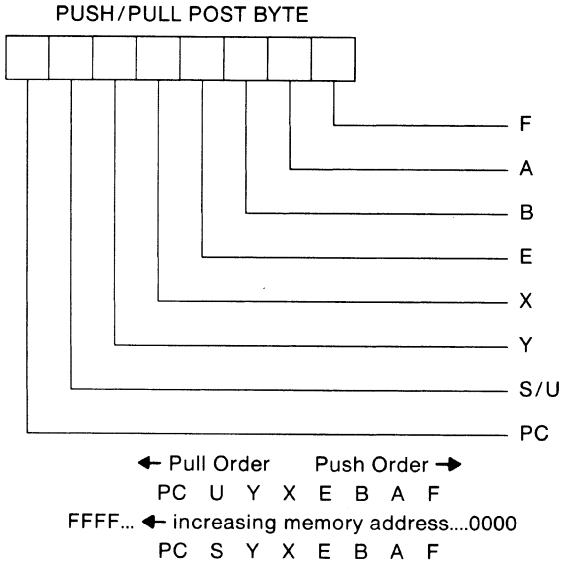
LDXEX TABLE,PC is load X with the effective address TABLE,PC. The value TABLE is temporarily added to the program counter to get an address. The X register is loaded with this address. Since program counter relative is a type of indexing, an additional level of indirection is available.

```
A6 9C —     LDAX     (CAT,PC)
EE 9C —     LDU      (DOG,PC)
```

GENERATING THE POST BYTE

A post byte follows some op codes to specify the basic type and variation of indexed addressing and/or registers involved.

- PSrS, PSrU, PLrS, PLrU
- The push instructions have the capability of pushing onto either the hardware stack (S) or user stack (U) any single register, or set of registers with a single instruction.
- The pull instructions have the same capability as the push instruction, in reverse order. The byte immediately following the push or pull opcode determines which register or registers are to be pushed or pulled. The actual PSrS/PLrS sequence is fixed; each bit defines a unique register to push or pull, as shown in below.



Example:

34 46 PSrS A,B,U

The opcode for PSrS is 34. Looking at the chart above, the post byte has a 1 for each register that is pushed which gives 01000110=46. First U is pushed, then B, then A. A PLrS A,B,U will pull the registers in the reverse order of storage, pulling A, then B, then U.

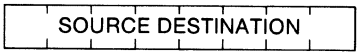
TFR/EXG

Within the 6809, any register may be transferred to or exchanged with another of like-size; i.e, 8-bit to 8-bit or 16-bit to 16-bit. Bits 4-7 of the post byte define the source register, while bits 0-3 represent the destination register. These are denoted as follows:

0000 - D	0101 - PC
0001 - X	1000 - A
0010 - Y	1001 - B
0011 - U	1010 - F
0100 - S	1011 - E

Note: All other combinations are undefined and INVALID.

TRANSFER/EXCHANGE POST BYTE



Examples:

1F 98 LDA B
1E 31 XCX U (same as 1E 13 XCU X)

LDA B is load register A from register B. The LDrr opcode is 1F. The source register is B which is represented by 1001 or 9, the destination of the load is A which is represented by 1000 or 8. So the LDA B instruction is encoded as 1F 98.

LDXE, LDYE, LDUE, LDSE

The LDrE (Load Register Effective) works by calculating the effective address used in an indexed instruction and storing that address value, rather than the data at the address, in a pointer register. This makes all the features of the internal addressing hardware available to the programmer. Some of the implications of this instruction are illustrated in Figure A-6.

The LDrE instruction also allows the user to access data in a position independent manner. For example:

```
        LDXEX       MSG1,PCR
        JSRL   UN   PDATA (Print message routine)
        •
        •
MSG1                'MESSAGE'
```

This sample program prints: 'MESSAGE'. By writing MSG1, PCR, the assembler computes the distance between the present address and MSG1. This result is placed as a constant into the LDXEX instruction which will be indexed from the PC value at the time of execution. No matter where the code is located, when it is executed, the computed offset from the PC will put the absolute address of MSG1 into the X pointer register. This code is totally position independent.

Instruction		Operation	Comments
30 07	LDXEX 7,X	X + 7 → X	Adds 5-bit constant 7 to X
30 89 05 00	LDXEX 500,X	X + 500 → X	Adds 16-bit constant 500 to X
31 A6	LDYEX A,Y	X + A → Y	Adds 8-bit A accumulator to Y
31 AB	LDYEX D,Y	Y + D → Y	Adds 16-bit D accumulator to Y
33 59	LDUEX - 7,U	U - 7 → U	Subtracts 7 from U
32 79	LDSEX - 7,S	S - 7 → S	Used to reserve area on stack
32 67	LDSEX 7,S	S + 7 → S	Used to "clean up" stack
30 05	LDXEX 5,S	S + 5 → X	Transfers as well as adds
30 1F	LDXEX -1,X	X - 1 → X	Decrement X

Figure A-6. Load Register Effective Examples

The post byte is a form of the indexed addressing mode described on the following page.

Indexed Addressing Post Byte

The indexed addressing post byte is looked up in a table. The post byte is in binary and a register code or 5-bit offset may have to be inserted into the binary opcode. Some of the post bytes are followed by additional bytes that contain an offset or address.

Examples:

```
30 1F      LDXEX   -1,X
AE 84      LDDX    0,X
AE AB      LDXX    D,Y
ED A1      STDX    ,Y + +
A5 B8 02   LDAX    (2,Y)
```

Type	Forms	Non Indirect				Indirect			
		Assembler Form	Postbyte OP Code	+ —	+ /	Assembler Form	Postbyte OP Code	+ —	+ /
Constant Offset From R (2's Complement Offsets)	No Offset	,R	1RR00100	0	0	(,R)	1RR10100	3	0
	5 Bit Offset	n,R	0RRnnnnn	1	0	defaults to 8-bit			
	8 Bit Offset	n, R	1RR01000	1	1	(n, R)	1RR11000	4	1
	16 Bit Offset	n, R	1RR01001	4	2	(n, R)	1RR11001	7	2
Accumulator Offset from R (2's Complement Offsets)	A Register Offset	A, R	1RR00110	1	0	(A, R)	1RR10110	4	0
	B Register Offset	B, R	1RR00101	1	0	(B, R)	1RR10101	4	0
	D Register Offset	D, R	1RR01011	4	0	(D, R)	1RR11011	7	0
Auto Increment/ Decrement R	Increment By 1	,R+	1RR00000	2	0	not allowed			
	Increment By 2	,R++	1RR00001	3	0	(,R++)	1RR10001	6	0
	Decrement By 1	,-R	1RR00010	2	0	not allowed			
	Decrement By 2	,--R	1RR00011	3	0	(,--R)	1RR10011	6	0
Constant Offsets From PC (2's Complement Offsets)	8 Bit Offset	n, PCR	1xx01100	1	1	(n, PCR)	1xx11100	4	1
	16 Bit Offset	n, PCR	1xx01101	5	2	(n, PCR)	1xx11101	8	2
Extended Indirect	16 Bit Address	—	—	—	—	(n)	10011111	5	2

R = X, Y, U or S
x = Don't Care

RR:
00=X
01=Y
10=U
11=S

$\pm$ and $\dagger$ indicate the number of additional cycles and bytes for the particular variation.

Figure A-7. Indexed Addressing Mode

LDXEX -1,X is load X effective with -1,X. LDXEX has opcode 30. -1,X is a 5-bit offset, from the table above, the post byte is 0RRnnnnn. RR=00 for register X, nnnnn = 11111 for -1. Thus the post byte is 00011111 = 1F.

LDDX 0,X is load X indexed with 0,X. LDDX has opcode AE. 0,X is no offset, from the table—the post byte is 1RR00100 with RR=00 for register X the post byte = 84.

LDAX (2,Y) is load A indexed indirect from 2,Y. The opcode for LDAX is A6. Looking in the indirect table the post byte for 5-bit offsets is defaulted to 8 bits and is 1RR11000. RR = 01 giving 10111000 = B8. The post byte is followed by the 8-bit offset 02.

6809 PROGRAMMING APPLICATIONS

Accesses to Data on the Stack

Access can be made to the stack without performing a push or a pull for each operation. This allows temporary storage on the stack, and data stored here can be manipulated almost as easily as if it were in a register.

```
        34 06   PSRS   D          Save data on the stack
         •       •
         •       •
         •       •
        E6 E4   LDBX   0,S        Recall data from the
         •       •                stack without pulling the
         •       •                stack
         •       •
        E7 E4   STBX   0,S        Alter data on the stack
         •       •
LOOP     •       •
         •       •
        6A 61   DCMX   1,S        Decrement counter on
                                  the stack
        26 F5    JPR   NE LOOP    Jump if counter not zero
         •       •
         •       •
         •       •
        32 62   LDSEX  2,S        Increment the stack
                                  pointer to clean data from
                                  the stack
```

Direct Addressing of Memory Where the Memory Block Moves

If you want to use direct addressing (i.e., refer to memory by its actual physical address) there is a problem if this memory block can move. One solution is to reassemble the program each time memory moves. Another solution is to read the page address of this memory and load this into the direct page register. No reassembling is required. Also the direct addressing mode can be used which requires only 2 bytes per instruction. If the memory block is larger than one page, the direct page register (register E) can be loaded several times during execution.

```
86 —    START  LDAI   PORT     Read the page address
                              of RAM
1F 8B          LDE    A        Load the direct page register
 •              •
 •              •
 •              •
97 05          STAS   05       Store in the fifth location of
                                the RAM page
```

Stepping Through Data or Tables

Point to the table with an index register. Load data from the table in the indexed mode with an automatic increment or decrement to step through the table.

```
A6 80   LDAX ,X+   Get the data pointed to by X, increment
                   pointer
A6 82   LDAX ,-X   Decrement X, get the data pointed to
                   by X
```

e.g. search table for values =5

```
         •
         •
         •
8E — —        LDXI   TABLE
A6 80 LOOP    LDAX   ,X+
81 05         CPAI   05
26 FA         JPR NE LOOP
         •
         •
         •
```

Identical Operations on Tables of Data

Place all manipulations of table data in subroutines. Use the indexed mode when referring to table data. Each time the subroutine is called, load the index register with the address of the table being accessed.

e.g. Double all values in a table that has the end of the table indicated by a negative value.

```
86 — —          LDXI   TABLE1
8D — —          JSR    TABLEDOUBLE
86 — —          LDXI   TABLE2
8D — —          JSR    TABLEDOUBLE
   •
   •
   •
A6 84  TABLEDOUBLE LDA 0,X          Get a table value
2D 05              JPR LT END       Jump if end of table
47                 SLAA             Multiply value by 2
A7 80              STA,X+           Store value
20 F7              JPR UN TABLEDOUBLE  Continue doubling table
39           END   RTS
```

Row-Column Tables

Tables can be accessed using row and column numbers. Store the table data linearly in memory. Store all the information in one row followed by all the information in the next row.

e.g. get data from location Row Q, Column R (for 1 word location, table with 256 items or less)

```
8E — —          LDXI     TABLE       Get table base address
86 —            LDAI     ROWQ        Get row of data item
C6 —            LDBI     COLUMNR     Get column of data item
BD — —          JS   UN  FETCH       Get data item from table

34 04 FETCH PSRS         B           Save column of data item
C6 —            LDBI     TABLEWIDTH  Get number of columns in table
3D              MU                   Get offset into memory of row
3A              ABX                  Get physical address of row
35 04           PLRS     B           Recall column of data item
A6 85           LDAX     B,X         Get data item from memory
39              RTS
```

Example:

	Column 0 Item No.	Column 1 In stock quantity	Column 2 Reorder when quantity is	
Actual Table	100	25	2	Row 0
	105	178	100	Row 1
	108	15	5	Row 2
Stored Table	STORED: 100, 25,2, 105, 178, 100, 108, 15,5			

To access data about item 108, we need to know its row, which is 2. Its in stock quantity is in column 1. This can be referred to as TABLE (2,1).

To get the in stock quantity of item 108:

```
8E — —   LDXI      TABLE   Get table base address
86 02    LDAI      2       Get row of item (row 2)
C6 01    LDBI      1       Get column number (column 1)
BD — —   JS    UN  FETCH   Get the item
•        •
•        •
•        •
5C       ICB               Point to reorder quantity
                           column (column 2)
BD — —   JS    UN  FETCH   Get the item
•        •
•        •
•        •
86 08    LDAI      8       Point to an equivalent item
5A       DCB               Point to its in stock quantity
BD — —   JS    UN  FETCH   See if it is in stock if the
                           other wasn't
```

Determining The Current Value Of The Stack

LDXEX 12,S obtains the value the stack had before an interrupt came. This can be used to determine what some independent user has set his stack to.

Storing Without Interruptions

An interrupt will be recognized at the end of an instruction. For example, when you are in the middle of storing multibyte information that can be updated by the NMI routine; if an NMI comes during a store, data can be changed, and an NMI can't be masked.

However, PSrS and PSrU allow storing of several registers (i.e., multiple bytes) in a single instruction that can't be interrupted.

Example:

D and X contain a 4 byte count that is incremented by the NMI routine. This count must be stored periodically.

D:X = 0000FFFF

If D is stored, an interrupt occurs, and then X is stored, 00000000 would be stored and D:X would now equal 00010000.

Instead set the register U to the store location and use PSrU D,X to store the count.

Calling A User Routine After A Breakpoint

After a JI has halted processing of a user routine, the program can call the next instruction as a subroutine.

```
35 7F PLrS      F,D,E,X,Y,U   Recall all register states at
                              time of interrupt
AD F1 JSX UN    (,S++)        Call as subroutine the
                              address that was in the
                              program counter at the time
                              of the interrupt, and remove
                              the address from the stack.
```

Saving Working Registers

To keep subroutines from changing registers being used by the calling routine, push all registers used by the subroutine in the subroutine using a multiple push instruction which requires only two bytes.

```
34 53  SUBROUTINE PSrS     A,X,U,F,  Save working registers
  •
  •
  •
86 05              LDAI     5         Set loop counter to 5
  •       Loop
  •
  •
  •
 4A                DCA                Decrement number of times through loop
26 —               JPR NE   LOOP      Jump if loop not complete
  •
  •
  •
35 53              PSrS     A,X,U,F   Recall working registers
 39                RTS                Return to calling routine
```

The U Register and The Users Stack

The U register can be used as an index register just like X and Y, and in fact usually uses one byte less per instruction than Y. However, the U register is also the User Stack pointer and is controlled entirely by the programmer. This allows passing of arguments to subroutines, or an additional pointer to temporary storage. The U register can be used as a pointer to a stack of global storage.

Block Moves

A block of data can easily be moved in memory using two index registers and auto increment indexing.

Example:

```
8E — —      LDXI   BLOCK      Get address of start of
                              block to move
C6 —        LDBI   LENGTH     Get length of block to be
                              moved
CE — —      LDUI   NEWBLOCK   Get address of start of
                              new block

A6 80 LOOP  LDAX   ,X+        Get byte from old block,
                              incr. pointer
A7 A0       STAX   ,Y+        Store byte in new block,
                              incr. pointer
5A          DCB               Decrement count of
                              bytes to move
26 F9       JPR NE LOOP       Jump if still some bytes
                              to move
```

CUTTING DOWN THE NUMBER OF BYTES USED

R Register

Instructions involving the Y register generally require one more opcode byte than similar instructions using the X or U register. Use the Y register less than X and U.

D Register

The D register is the A and B registers combined. If both A and B registers need to be loaded or stored; load or store the D register instead if possible. This cuts down on the number of bytes used.

Direct Addressing

When memory operations tend to be in the same page, set the direct page register to that page. Then replace direct addressing instructions with short direct to save a byte each time.

Short Relative Addressing

Relative jumps of plus 127 or minus 128 bytes require 1 less byte than longer relative jumps or extended addressing jumps. Locating frequently called routines near calling routines saves bytes.

Automatic Increment and Decrement

Pointers to memory can be automatically incremented or decremented in the indexed mode of addressing.

Obtaining Data From An Address On The Stack

```
PLrS          LDUEX (,S++)   7889   23
        =                    788A   45

LDUX 0,U                     F000   78
                             F001   89
```

Example: S = F000

LDUEX (S,++) gets the address at S, increments S to F002 (takes 7889 off stack) and gets the data stored at address 7889, which is 2345. So S = F002 and U = 2345.

Pulling The Stack and Returning

If registers are saved at subroutine entry and then pulled just before return, the pulling and return can be combined.

```
PLRS   A,X,Y   =   PLRS A,X,Y,PC
RTS
```

APPENDIX B
The 6809 Instruction Set

This appendix contains general information about the 6809 instruction set.

THE 6809 INSTRUCTION SET

Figures B-1 through B-5 present the 6809 instruction set with STD mnemonics and functionally grouped by: Register, Memory, Program Address Control, Machine, Register Pair, Input/Output and Restart instructions.

The flag status nomenclature is as follows: Lower case letters are variable, upper case followed by 0 or 1 indicate absolute condition. Only the affected flags are tabulated.

Figures B-13 and B-14 detail all possible post bytes that can follow an op code. These can also be generated from figures B-4 through B-12. The information presented in Figures B-4 through B-12 is available from Pro-Log as a shirt-pocket programming aid card, document # 108767.

Figures B-15 and B-16 define the notations used in Figures B-4 through B-12. Following figure B-16 are definitions and specifications of the 6809's instruction set.

STD INSTRUCTION MNEMONICS

The STD Instruction Mnemonics are a standard set of processor instruction abbreviations suitable for use as an assembly language for writing programs.

These mnemonics are standard in that they do not change but keep the same meaning regardless of the processor they are applied to. They are also standard in that they are derived from a set of easily understood rules.

The instruction mnemonic is an abbreviated action statement containing an operator, a locator and a qualifier plus a supplemental and separate modifier.

1. The operator is a unique two letter abbreviation that suggests the action.
2. The locator follows the operator and designates the operand or data to be operated on. Instructions without operands ignore the locator.
3. The qualifier states the addressing mode or provides further qualifying information for compound instructions.
4. The modifier carries detailed support information; labels, conditions, addressing and data.

The operator, locator and qualifier letters are strung together to form the instruction mnemonic. The modifier, when needed, stands alone either in its own separate column or separated by spaces or additional lines in written text.

	Operator	Locator	Qualifier	Modifier	Instruction Description
RTS	RT	S			Return from Subroutine
CLA	CL	A			Clear A
LDAD	LD	A	D		Load A Direct
LDA B	LD	A		B	Load A with B
LDAN (BC)	LD	A	N	(BC)	Load A indirect using BC as an Address Pointer
JS (LABEL)	JS			(LABEL)	Jump to Subroutine Located at (LABEL)

Figure B-1. Examples of Instruction Mnemonic Structure

6809 MNEMONICS		STD MNEMONICS	
ABX		ADXB	
ADC		AC	
ADD		AD	
AND		AN	
ASLr		SLrA	
ASRr		SRrA	
BCC	LBCC	JPR C0	JPRL C0
BCS	LBCS	JPR C1	JPRL C1
BEQ	LBEQ	JPR EQ	JPRL EQ
BGE	LBGE	JPR GE	JPRL GE
BGT	LBGT	JPR GT	JPRL GT
BHI	LBHI	JPR H	JPRL H
BHS	LBHS	JPR NL	JPRL NL
BIT		ANrT	
BLE	LBLE	JPR LE	JPRL LE
BLO	LBLO	JPR L	JPRL L
BLS	LBLS	JPR NH	JPRL NH
BLT	LBLT	JPR LT	JPRL LT
BMI	LBMI	JPR N	JPRL N
BNE	LBNE	JPR NE	JPRL NE
BPL	LBPL	JPR P	JPRL P
BRA	LBRA	JPR UN	JPRL UN
BRN	LBRN	JPR + 1	JPR + 2
BSR	LBSR	JSR UN	JSRL UN
BVC	LBVC	JPR NV	JPRL NV
BVS	LBVS	JPR V	JPRL V
CLR		CL	
CMP		CP	
COMr		CMrL	
CWAI		HLT C	
DAA		AJAD	
DEC		DC	
EOR		XR	
EXG		XCR	
INC		IC	
JMP		JP UN	
JSR		JS UN	
LEAr		LDrE	
LSLr		SLrL	
LSRr		SRrL	
MUL		MU	
NEGr		CMrA	
PSH		PSr	
PUL		PLr	
ROL		RL	
ROR		RR	
SBC		SC	
SEX		AJW	
SUB		SU	
SWI		JI	
SYNC		HLT Z	
TFR r,r		LDr2 r1	
TSTr		CPrZ	

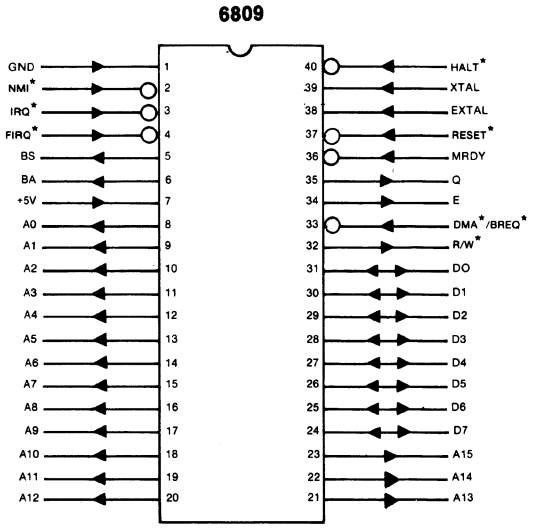
Figure B-2. 6809-STD Mnemonic Equivalency

6809	STD
CCR or Condition code register	F or Flag register
DP or Direct Page register	E or Direct Page register
BR or Branch	JPR or jump relative
# or Immediate value follows	I or Immediate addressing mode
Direct	Short Direct
Extended	Direct

Figure B-3. 6809-STD Terminology Equivalency

PRO-LOG® CORPORATION

6809 Processor Programming Aids



INSTR.	MODI-FIER		S	D	I	X'	FLAG	STATUS		DESCRIPTION		
LDA		—	96	B6	86	A6	s	z	V0	M→A		LOAD REGISTER A
LDB		—	D6	F6	C6	E6	s	z	V0	M→B		LOAD REGISTER B
LDD		—	DC	FC	CC	EC	s	z	V0	M:M+1→D		LOAD REGISTER D
LDS		—	10DE	10FE	10CE	10EE	s	z	V0	M:M+1→S		LOAD REGISTER S
LDU		—	DE	FE	CE	EE	s	z	V0	M:M+1→U		LOAD REGISTER U
LDX		—	9E	BE	8E	AE	s	z	V0	M:M+1→X		LOAD REGISTER X
LDY		—	109E	10BE	108E	10AE	s	z	V0	M:M+1→Y		LOAD REGISTER Y
STA		—	97	B7	—	A7	s	z	V0	A→M		STORE REGISTER A
STB		—	D7	F7	—	E7	s	z	V0	B→M		STORE REGISTER B
STD		—	DD	FD	—	ED	s	z	V0	D→M:M+1		STORE REGISTER D
STS		—	10DF	10FF	—	10EF	s	z	V0	S→M:M+1		STORE REGISTER S
STU		—	DF	FF	—	EF	s	z	V0	U→M:M+1		STORE REGISTER U
STX		—	9F	BF	—	AF	s	z	V0	X→M:M+1		STORE REGISTER X
STY		—	109F	10BF	—	10AF	s	z	V0	Y→M:M+1		STORE REGISTER Y
LDSE		—	—	—	—	32				EA→S		LOAD EFFECTIVE ADDRESS INTO S
LDUE		—	—	—	—	33				EA→U		LOAD EFFECTIVE ADDRESS INTO U
LDXE		—	—	—	—	30		z		EA→X		LOAD EFFECTIVE ADDRESS INTO X
LDYE		—	—	—	—	31		z		EA→Y		LOAD EFFECTIVE ADDRESS INTO Y
PSRS	1	34	—	—	—	—				PUSH ON S STACK		PUSH REGISTERS ON S STACK
PSRU	1	36	—	—	—	—				PUSH ON U STACK		PUSH REGISTERS ON U STACK
PLRS	1	35	—	—	—	—				PULL FROM S STACK		PULL REGISTERS FROM S STACK
PLRU	1	37	—	—	—	—				PULL FROM U STACK		PULL REGISTERS FROM U STACK
XCR	R1,R2	1E	—	—	—	—				R1↔R2	1	EXCHANGE REGISTERS
LDR2	R1	1F	—	—	—	—				R1→R2	1	LOAD REGISTER FROM REGISTER

Figure B-4. Load and Store Instructions

RELATIVE JUMP INSTRUCTIONS (Flag Status Unchanged)

INSTR.	MODIFIER		DESCRIPTION	
JPR	C0	24	JUMP C=0	JUMP RELATIVE IF CARRY CLEAR
JPRL	C0	1024	JUMP C=0	LONG JUMP RELATIVE IF CARRY CLEAR
JPR	C1	25	JUMP C=1	JUMP RELATIVE IF CARRY SET
JPRL	C1	1025	JUMP C=1	LONG JUMP RELATIVE IF CARRY SET
JPR	EQ	27	JUMP Z=0	JUMP RELATIVE EQUAL TO ZERO
JPRL	EQ	1027	JUMP Z=0	LONG JUMP RELATIVE EQUAL TO ZERO
JPR	GE	2C	JUMP ≥ ZERO	JUMP RELATIVE ≥ ZERO
JPRL	GE	102C	JUMP ≥ ZERO	LONG JUMP RELATIVE ≥ ZERO
JPR	GT	2E	JUMP > ZERO	JUMP RELATIVE > ZERO
JPRL	GT	102E	JUMP > ZERO	LONG JUMP RELATIVE > ZERO
JPR	H	22	JUMP HIGHER	JUMP RELATIVE HIGHER THAN
JPRL	H	1022	JUMP HIGHER	LONG JUMP RELATIVE HIGHER THAN
JPR	L	25	JUMP LOWER	JUMP RELATIVE LOWER THAN
JPRL	L	1025	JUMP LOWER	LONG JUMP RELATIVE LOWER THAN
JPR	LE	2F	JUMP ≤ ZERO	JUMP RELATIVE ≤ ZERO
JPRL	LE	102F	JUMP ≤ ZERO	LONG JUMP RELATIVE ≤ ZERO
JPR	LT	2D	JUMP < ZERO	JUMP RELATIVE < ZERO
JPRL	LT	102D	JUMP < ZERO	LONG JUMP RELATIVE < ZERO
JPR	N	2B	JUMP NEGATIVE	JUMP RELATIVE NEGATIVE
JPRL	N	102B	JUMP NEGATIVE	LONG JUMP RELATIVE NEGATIVE
JPR	NE	26	JUMP Z≠0	JUMP RELATIVE NOT EQUAL ZERO
JPRL	NE	1026	JUMP Z≠0	LONG JUMP RELATIVE NOT EQUAL ZERO
JPR	NH	23	JUMP LOWER OR SAME	JUMP RELATIVE NOT HIGHER THAN
JPRL	NH	1023	JUMP LOWER OR SAME	LONG JUMP RELATIVE NOT HIGHER THAN
JPR	NL	24	JUMP HIGHER OR SAME	JUMP RELATIVE NOT LOWER THAN
JPRL	NL	1024	JUMP HIGHER OR SAME	LONG JUMP RELATIVE NOT LOWER THAN
JPR	NV	28	JUMP V=0	JUMP RELATIVE OVERFLOW CLEAR
JPRL	NV	1028	JUMP V=0	LONG JUMP RELATIVE OVERFLOW CLEAR
JPR	P	2A	JUMP POSITIVE	JUMP RELATIVE POSITIVE
JPRL	P	102A	JUMP POSITIVE	LONG JUMP RELATIVE POSITIVE
JPR	UN	20	JUMP ALWAYS	JUMP RELATIVE UNCONDITIONALLY
JPRL	UN	16	JUMP ALWAYS	LONG JUMP RELATIVE UNCONDITIONALLY
JPR	V	29	JUMP V=1	JUMP RELATIVE IF OVERFLOW SET
JPRL	V	1029	JUMP V=1	LONG JUMP RELATIVE IF OVERFLOW SET
JPR	+1	21	JUMP NEVER	JUMP RELATIVE 1 (SKIP 1)
JPR	+2	1021	JUMP NEVER	LONG JUMP RELATIVE 2 (SKIP 2)
JSR		8D	JUMP TO SUBROUTINE	JUMP RELATIVE TO SUBROUTINE
JSRL		17	JUMP TO SUBROUTINE	LONG JUMP RELATIVE TO SUBROUTINE

ARITHMETIC INSTRUCTIONS

INSTR.	MODIFIER		S	D	I	X'	FLAG STATUS					DESCRIPTION	
ADA		—	9B	BB	8B	AB	d	s	z	v	c	A+M→A	ADD TO A
ADB		—	DB	FB	CB	EB	d	s	z	v	c	B+M→B	ADD TO B
ADD		—	D3	F3	C3	E3	d	s	z	v	c	D+M:M+1→D	ADD TO D
ACA		—	99	B9	89	A9	d	s	z	v	c	A+M+C→A	ADD TO A WITH CARRY
ACB		—	D9	F9	C9	E9	d	s	z	v	c	B+M+C→B	ADD TO B WITH CARRY
SUA		—	90	B0	80	A0	?	s	z	v	c	A-M→A	SUBTRACT FROM A
SUB		—	D0	F0	C0	E0	?	s	z	v	c	B-M→B	SUBTRACT FROM B
SUD		—	93	B3	83	A3		s	z	v	c	D-M:M+1→D	SUBTRACT FROM D
SCA		—	92	B2	82	A2	?	s	z	v	c	A-M-C→A	SUBTRACT FROM A WITH CARRY
SCB		—	D2	F2	C2	E2	?	s	z	v	c	B-M-C→B	SUBTRACT FROM B WITH CARRY
ADX	B	3A	—	—	—	—						B+X→X (UNSIGNED)	ADD B TO X
MU		3D	—	—	—	—			z		3	AxB→D (UNSIGNED)	MULTIPLY A AND B
ICA		4C	—	—	—	—		s	z	v		A+1→A	INCREMENT A
ICB		5C	—	—	—	—		s	z	v		B+1→B	INCREMENT B
ICM		—	0C	7C	—	6C		s	z	v		M+1→M	INCREMENT MEMORY
DCA		4A	—	—	—	—		s	z	v		A-1→A	DECREMENT A
DCB		5A	—	—	—	—		s	z	v		B-1→B	DECREMENT B
DCM		—	0A	7A	—	6A		s	z	v		M-1→M	DECREMENT MEMORY
CLA		4F	—	—	—	—		S0	Z1	V0	C0	0→A	CLEAR A
CLB		5F	—	—	—	—		S0	Z1	V0	C0	0→B	CLEAR B
CLM		—	0F	7F	—	6F		S0	Z1	V0	C0	0→M	CLEAR MEMORY
AJAD		19	—	—	—	—		s	z	V0	c	DECIMAL ADJUST A	DECIMAL ADJUST A

Figure B-5. Relative Jump, Arithmetic Instructions

PROGRAM ADDRESS CONTROL

INSTR.	MODIFIER		S	D	I	X'	FLAG STATUS	DESCRIPTION	
JP	UN	—	0E	7E	—	6E		EA→PC	JUMP UNCONDITIONALLY
JS	UN	—	9D	BD	—	AD		JUMP TO SUBROUTINE	JUMP TO SUBR. UNCONDITIONALLY
RTS		39	—	—	—	—		RETURN FROM SUBROUTINE	RETURN FROM SUBROUTINE
JI		3F	—	—	—	—		SOFTWARE INTERRUPT 1	JUMP TO INTERRUPT 1
JI2		103F	—	—	—	—		SOFTWARE INTERRUPT 2	JUMP TO INTERRUPT 2
JI3		113F	—	—	—	—		SOFTWARE INTERRUPT 3	JUMP TO INTERRUPT 3
RTI		3B	—	—	—	—	2	RETURN FROM INTERRUPT	RETURN FROM INTERRUPT
HLT	C	3C	—	—	—	—	C1	cc•m→cc, STACK, HALT	STACK, CLEAR FLAGS, HALT
HLT	Z	13	—	—	—	—		SYNCHRONIZE TO INTERRUPT	HALT AND TRI-STATE
NOP		12	—	—	—	—		NO OPERATION	NO OPERATION

LOGICAL INSTRUCTIONS

INSTR.		S	D	I	X'	FLAG STATUS					DESCRIPTION	
ANA	—	94	B4	84	A4		s	z	V0		A•M→A	AND WITH A
ANB	—	D4	F4	C4	E4		s	z	V0		B•M→B	AND WITH B
ANF	—	—	—	1C	—					2	F•M→F	AND WITH F
ORA	—	9A	BA	8A	AA		s	z	V0		A+M→A	OR WITH A
ORB	—	DA	FA	CA	EA		s	z	V0		B+M→B	OR WITH B
ORF	—	—	—	1A	—					2	F+M→F	OR WITH F
XRA	—	98	B8	88	A8		s	z	V0		A⊕M→A	EXCLUSIVE OR WITH A
XRB	—	D8	F8	C8	E8		s	z	V0		B⊕M→B	EXCLUSIVE OR WITH B
CPA	—	91	B1	81	A1	?	s	z	v	c	COMPARE A TO M	COMPARE WITH A
CPB	—	D1	F1	C1	E1	?	s	z	v	c	COMPARE B TO M	COMPARE WITH B
CPD	—	1093	10B3	1083	10A3		s	z	v	c	COMPARE D TO M:M+1	COMPARE WITH D
CPS	—	119C	11BC	118C	11AC		s	z	v	c	COMPARE S TO M:M+1	COMPARE WITH S
CPU	—	1193	11B3	1183	11A3		s	z	v	c	COMPARE U TO M:M+1	COMPARE WITH U
CPX	—	9C	BC	8C	AC		s	z	v	c	COMPARE X TO M:M+1	COMPARE WITH X
CPY	—	109C	10BC	108C	10AC		s	z	v	c	COMPARE Y TO M:M+1	COMPARE WITH Y
ANAT	—	95	B5	85	A5		s	z	V0		BIT TEST A (M•A)	AND WITH A TEST
ANBT	—	D5	F5	C5	E5		s	z	V0		BIT TEST B (M•A)	AND WITH B TEST
CMAL	43	—	—	—	—		s	z	V0	C1	A*→A	COMPLEMENT 1's (LOGICAL) A
CMBL	53	—	—	—	—		s	z	V0	C1	B*→B	COMPLEMENT 1's (LOGICAL) B
CMML	—	03	73	—	63		s	z	V0	C1	M*→M	COMPLEMENT 1's (LOGICAL) M
CMAA	40	—	—	—	—	?	s	z	v	c	A*+1→A	COMPLEMENT 2's (ARITHMETIC) A
CMBA	50	—	—	—	—	?	s	z	v	c	B*+1→B	COMPLEMENT 2's (ARITHMETIC) B
CMMA	—	00	70	—	60	?	s	z	v	c	M*+1→M	COMPLEMENT 2's (ARITHMETIC) M
AJW	1D	—	—	—	—		s	z	V0		SIGN EXTEND B INTO A	ADJUST WORD
CPAZ	4D	—	—	—	—		s	z	V0		TEST A	COMPARE A WITH 0
CPBZ	5D	—	—	—	—		s	z	V0		TEST B	COMPARE B WITH 0
CPMZ	—	0D	7D	—	6D		s	z	V0		TEST M	COMPARE M WITH 0
RLA	49	—	—	—	—		s	z	v	c	A	ROTATE LEFT A
RLB	59	—	—	—	—		s	z	v	c	B	ROTATE LEFT B
RLM	—	09	79	—	69		s	z	v	c	M	ROTATE LEFT M
RRA	46	—	—	—	—		s	z		c	A	ROTATE RIGHT A
RRB	56	—	—	—	—		s	z		c	B	ROTATE RIGHT B
RRM	—	06	76	—	66		s	z		c	M	ROTATE RIGHT M
SLAA	48	—	—	—	—	?	s	z	v	c	A	SHIFT LEFT ARITHMETIC A
SLBA	58	—	—	—	—	?	s	z	v	c	B	SHIFT LEFT ARITHMETIC B
SLMA	—	08	78	—	68	?	s	z	v	c	M	SHIFT LEFT ARITHMETIC M
SRAA	47	—	—	—	—	?	s	z		c	A	SHIFT RIGHT ARITHMETIC A
SRBA	57	—	—	—	—	?	s	z		c	B	SHIFT RIGHT ARITHMETIC B
SRMA	—	07	77	—	67	?	s	z		c	M	SHIFT RIGHT ARITHMETIC M
SRAL	44	—	—	—	—		S0	z		c	A	SHIFT RIGHT LOGICAL A
SRBL	54	—	—	—	—		S0	z		c	B	SHIFT RIGHT LOGICAL B
SRML	—	04	74	—	64		S0	z		c	M	SHIFT RIGHT LOGICAL M

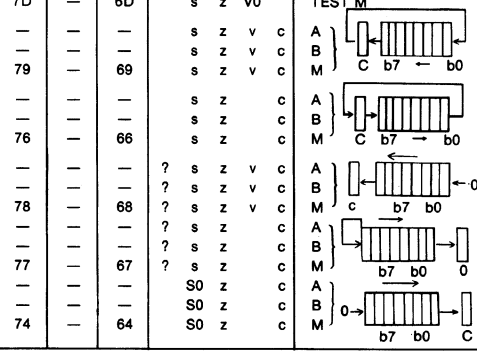


Figure B-6. Program Address Control, Logical Instructions

NOTES:

1. OP CODE IS FOLLOWED BY A POST BYTE
2. FLAG SET AS A DIRECT RESULT OF THE INSTRUCTION
3. SPECIAL CASE — CARRY SET IF b7 is SET
4. EA = EFFECTIVE ADDRESS
5. 5(6) MEANS: 5 CYCLES IF JUMP NOT TAKEN, 6 CYCLES IF TAKEN

ADDRESING MODE QUALIFIERS

	INHERENT
S	SHORT (UPPER BYTE FOR DIRECT ADDRESSING COMES FROM E REGISTER)
D	DIRECT
X	INDEXED
I	IMMEDIATE
R	RELATIVE (ONE BYTE OFFSET)
RL	LONG RELATIVE (TWO BYTE OFFSET)

FLAG REGISTER

d	DECIMAL (HALF) CARRY
s	SIGN
z	ZERO
v	OVERFLOW
c	CARRY/BORROW

FLAG STATUS

	IF FLAG REGISTER IS DISPLAYED, THE FLAG CHANGES ACCORDING TO OPERATION RESULT. IF FOLLOWED BY 0 OR 1, FLAG IS SET TO THAT VALUE
?	INDICATES FLAG VALUE IS UNDEFINED

FORWARD RELATIVE OFFSET

ADDRESS FOR 0 THRU 127 COUNTS

0	00	16	10	32	20	48	30	64	40	80	50	96	60	112	70
1	01	17	11	33	21	49	31	65	41	81	51	97	61	113	71
2	02	18	12	34	22	50	32	66	42	82	52	98	62	114	72
3	03	19	13	35	23	51	33	67	43	83	53	99	63	115	73
4	04	20	14	36	24	52	34	68	44	84	54	100	64	116	74
5	05	21	15	37	25	53	35	69	45	85	55	101	65	117	75
6	06	22	16	38	26	54	36	70	46	86	56	102	66	118	76
7	07	23	17	39	27	55	37	71	47	87	57	103	67	119	77
8	08	24	18	40	28	56	38	72	48	88	58	104	68	120	78
9	09	25	19	41	29	57	39	73	49	89	59	105	69	121	79
10	0A	26	1A	42	2A	58	3A	74	4A	90	5A	106	6A	122	7A
11	0B	27	1B	43	2B	59	3B	75	4B	91	5B	107	6B	123	7B
12	0C	28	1C	44	2C	60	3C	76	4C	92	5C	108	6C	124	7C
13	0D	29	1D	45	2D	61	3D	77	4D	93	5D	109	6D	125	7D
14	0E	30	1E	46	2E	62	3E	78	4E	94	5E	110	6E	126	7E
15	0F	31	1F	47	2F	63	3F	79	4F	95	5F	111	6F	127	7F

COUNT ZERO AT SECOND ADDRESS AFTER JUMP MNEMONIC

Figure B-7. Forward Relative Offset

BACKWARD RELATIVE OFFSET

ADDRESS FOR 1 THRU 128 COUNTS

1	FF	17	EF	33	DF	49	CF	65	BF	81	AF	97	9F	113	8F
2	FE	18	EE	34	DE	50	CE	66	BE	82	AE	98	9E	114	8E
3	FD	19	ED	35	DD	51	CD	67	BD	83	AD	99	9D	115	8D
4	FC	20	EC	36	DC	52	CC	68	BC	84	AC	100	9C	116	8C
5	FB	21	EB	37	DB	53	CB	69	BB	85	AB	101	9B	117	8B
6	FA	22	EA	38	DA	54	CA	70	BA	86	AA	102	9A	118	8A
7	F9	23	E9	39	D9	55	C9	71	B9	87	A9	103	99	119	89
8	F8	24	E8	40	D8	56	C8	72	B8	88	A8	104	98	120	88
9	F7	25	E7	41	D7	57	C7	73	B7	89	A7	105	97	121	87
10	F6	26	E6	42	D6	58	C6	74	B6	90	A6	106	96	122	86
11	F5	27	E5	43	D5	59	C5	75	B5	91	A5	107	95	123	85
12	F4	28	E4	44	D4	60	C4	76	B4	92	A4	108	94	124	84
13	F3	29	E3	45	D3	61	C3	77	B3	93	A3	109	93	125	83
14	F2	30	E2	46	D2	62	C2	78	B2	94	A2	110	92	126	82
15	F1	31	E1	47	D1	63	C1	79	B1	95	A1	111	91	127	81
16	F0	32	E0	48	D0	64	C0	80	B0	96	A0	112	90	128	80

COUNT ONE AT FIRST ADDRESS AFTER JUMP MNEMONIC

SPECIAL OPERATIONS		
OPERATION	**INSTRUCTION**	**OP CODES**
set carry	ORFI 01	1A 01
clear carry	ANFI FE	IC FE
increment S	LDSEX I,S	32 61
increment U	LDUEX I,U	33 41
increment X	LDXEX I,X	30 01
increment Y	LDYEX I,Y	31 21
decrement S	LDSEX -I,S	32 7F
decrement U	LDUEX -I,U	33 5F
decrement X	LDXEX -I,X	30 1F
decrement Y	LDYEX -I,Y	31 3F
load E	LDAI nn	86 nn
—	LDDE A	1F 8B

Figure B-8. Backward Relative Offset, Special Operations

HOW TO USE THE TABLES

CONVERSION TO DECIMAL: Find the decimal weights for corresponding hexadecimal characters beginning with the least significant character. The sum of the decimal weight is the decimal value of the hexadecimal number.

CONVERSION TO HEXADECIMAL: Find the highest decimal value in the table which is lower than or equal to the decimal number to be converted. The corresponding hexadecimal character is the most significant character. Subtract the decimal value found from the decimal number to be converted. Repeat the process with the difference, to find subsequent hexadecimal characters.

HEXADECIMAL AND DECIMAL CONVERSION

15 BYTE 8						7 BYTE 0			
15 CHAR 12			11 CHAR 8			7 CHAR 4		3 CHAR 0	
HEX		DEC	HEX		DEC	HEX	DEC	HEX	DEC
0		0	0		0	0	0	0	0
1	4	096	1		256	1	16	1	1
2	8	192	2		512	2	32	2	2
3	12	288	3		768	3	48	3	3
4	16	384	4	1	024	4	64	4	4
5	20	480	5	1	280	5	80	5	5
6	24	576	6	1	536	6	96	6	6
7	28	672	7	1	792	7	112	7	7
8	32	768	8	2	048	8	128	8	8
9	36	864	9	2	304	9	144	9	9
A	40	960	A	2	560	A	160	A	10
B	45	056	B	2	816	B	176	B	11
C	49	152	C	3	072	C	192	C	12
D	53	248	D	3	328	D	208	D	13
E	57	344	E	3	584	E	224	E	14
F	61	440	F	3	840	F	240	F	15

POWERS OF TWO

2^{η}	η
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1,024	10
2,048	11
4,096	12
8,192	13
16,384	14
32,768	15
65,536	16
131,072	17
262,144	18
524,288	19
1,048,576	20

Figure B-9. Stacking Order, Hexadecimal and Decimal Conversion

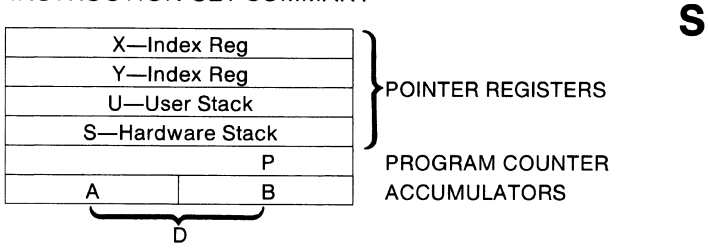
ASCII CHARACTER SET (7-BIT CODE)

LS CHAR / MS CHAR	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111
0 000	NUL	SOH	STX	ETC	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1 001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	VS
2 010	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3 011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4 100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5 101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑	—
6 110	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7 111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

PRO-LOG® CORPORATION

6809 Microprocessor Instruction Set Summary

INSTRUCTION SET SUMMARY



6809 STACKING ORDER

	PULL ORDER ↓	
	F	6809 VECTORS
INCREASING MEMORY ↓	A	FFFE Restart
	B	FFFC NMI
	E	FFFA JI
	X Hi	FFF8 IRQ
	X Lo	FFF6 FIRQ
	Y Hi	FFF4 JI2
	Y Lo	FFF2 JI3
	U/S Hi	FFF0 Reserved
	U/S Lo	
	P Hi	
	P Lo	
	↑ PUSH ORDER	

SIMPLE CONDITIONAL JUMPS

Condition	*Complement*
JPR EQ	JPR NE
JPR N	JPR P
JPR C1	JPR C0
JPR V	JPR NV

SIGNED CONDITIONAL JUMPS

Condition	*Complement*
JPR GT	JPR LE
JPR GE	JPR LT
JPR EQ	JPR NE
JPR LE	JPR GT
JPR LT	JPR GE

UNSIGNED CONDITIONAL JUMPS

Condition	*Complement*
JPR H	JPR NH
JPR NL	JPR L
JPR EQ	JPR NE
JPR NH	JPR H
JPR L	JPR NL

INDEXED ADDRESSING
POST BYTE REGISTER
BIT ASSIGNMENTS

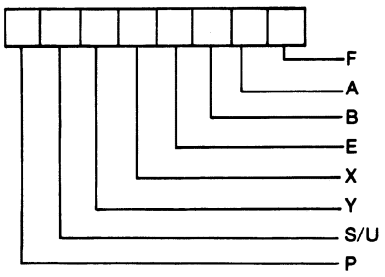
POST BYTE REGISTER BIT								INDEXED ADDRESSING MODE
7	6	5	4	3	2	1	0	
0	X	X	X	X	X	X	X	EA = , R ± 4 BIT OFFSET
1	X	X	0	0	0	0	0	, R +
1	X	X	X	0	0	0	1	, R + +
1	X	X	0	0	0	1	0	, - R
1	X	X	X	0	0	1	1	, - - R
1	X	X	X	0	1	0	0	EA = , R ± 0 OFFSET
1	X	X	X	0	1	0	1	EA = , R ± ACCB OFFSET
1	X	X	X	0	1	1	0	EA = , R ± ACCA OFFSET
1	X	X	X	1	0	0	0	EA = , R ± 7 BIT OFFSET
1	X	X	X	1	0	0	1	EA = , R ± 15 BIT OFFSET
1	X	X	X	1	0	1	1	EA = , R ± D OFFSET
1	X	X	X	1	1	0	0	EA = , P ± 7 BIT OFFSET
1	X	X	X	1	1	0	1	EA = , P ± 15 BIT OFFSET
1	X	X	1	1	1	1	1	EA = , ADDRESS
								ADDRESSING MODE FIELD

REGISTER FIELD — I FIELD

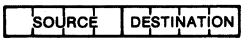
00:R = X
01:R = Y
10:R = U
11:R = S

FOR B7 = 1: INDIRECT
FOR B7 = 0: SIGN BIT

PUSH/PULL POST BYTE



TRANSFER/EXCHANGE POST BYTE



REGISTER FIELD

0000 = D (A:B)	1000 = A
0001 = X	1001 = B
0010 = Y	1010 = F
0011 = U	1011 = E
0100 = S	
0101 = P	

Figure B-10. ASCII Set, Indexed Post Byte Bit Assignments

OP	MNEM	MODE	~	#	OP	MNEM	MODE	~	#	OP	MNEM	MODE	~	#
00	CMMA	S	6	2	21	JPR+1		3	2	39	RTS		5	1
03	CMML	S	6	2	22	JPR	H	3	2	3A	ADX B		3	1
04	SRML	S	6	2	23	JPR	NH	3	2	3B	RTI		6/15	1
06	RRM	S	6	2	24	JPR	NL	3	2	3C	HLT C		21	2
07	SRMA	S	6	2	25	JPR	C1	3	2	3D	MU		11	1
08	SLMA	S	6	2	26	JPR	NE	3	2	3F	JI		19	1
09	RLM	S	6	2	27	JPR	EQ	3	2	40	CMAA		2	1
0A	DCM	S	6	2	28	JPR	NV	3	2	43	CMAL		2	1
0C	ICM	S	6	2	29	JPR	V	3	2	44	SRAL		2	1
0D	CPMZ	S	6	2	2A	JPR	P	3	2	46	RRA		2	1
0E	JP	S	3	2	2B	JPR	N	3	2	47	SRAA		2	1
0F	CLM	S	6	2	2C	JPR	GE	3	2	48	SLAA		2	1
12	NOP		2	1	2D	JPR	LT	3	2	49	RLA		2	1
13	HLT Z		2	1	2E	JPR	GT	3	2	4A	DCA		2	1
16	JPRL		5	3	2F	JPR	LE	3	2	4C	ICA		2	1
17	JSRL		9	3	30	LDXE		4	2	4D	CPAZ		2	1
19	AJAD		2	1	31	LDYE		4	2	4F	CLA		2	1
1A	ORF	I	3	2	32	LDSE		4	2	50	CMBA		2	1
1C	ANF	I	3	2	33	LDUE		4	2	53	CMBL		2	1
1D	AJW		2	1	34	PSRS		5	2	54	SRBL		2	1
1E	XCR		8	2	35	PLRS		5	2	56	RRB		2	1
1F	LDR2		7	2	36	PSRU		5	2	57	SRBA		2	1
20	JPR		3	2	37	PLRU		5	2	58	SLBA		2	1

OP	MNEM	MODE	~	#	OP	MNEM	MODE	~	#	OP	MNEM	MODE	~	#
59	RLB		2	1	79	RLM		7	3	93	SUD	S	6	2
5A	DCB		2	1	7A	DCM		7	3	94	ANA	S	4	2
5C	ICB		2	1	7C	ICM		7	3	95	ANAT	S	4	2
5D	CPBZ		2	1	7D	CPMZ		7	3	96	LDA	S	4	2
5F	CLB		2	1	7E	JP		4	3	97	STA	S	4	2
60	CMMA	X	6	2	7F	CLM		7	3	98	XRA	S	4	2
63	CMML	X	6	2	80	SUA	I	2	2	99	ACA	S	4	2
64	SRML	X	6	2	81	CPA	I	2	2	9A	ORA	S	4	2
66	RRM	X	6	2	82	SCA	I	2	2	9B	ADA	S	4	2
67	SRMA	X	6	2	83	SUD	I	4	3	9C	CPX	S	6	2
68	SLMA	X	6	2	84	ANA	I	2	2	9D	JS	S	7	2
69	RLM	X	6	2	85	ANAT	I	2	2	9E	LDX	S	5	2
6A	DCM	X	6	2	86	LDA	I	2	2	9F	STX	S	5	2
6C	ICM	X	6	2	88	XRA	I	2	2	A0	SUA	X	4	2
6D	CPMZ	X	6	2	89	ACA	I	2	2	A1	CPA	X	4	2
6E	JP	X	3	2	8A	ORA	I	2	2	A2	SCA	X	4	2
6F	CLM	X	6	2	8B	ADA	I	2	2	A3	SUD	X	6	2
70	CMMA		7	3	8C	CPX	I	4	3	A4	ANA	X	4	2
73	CMML		7	3	8D	JSR		7	2	A5	ANAT	X	4	2
74	SRML		7	3	8E	LDX	I	3	3	A6	LDA	X	4	2
76	RRM		7	3	90	SUA	S	4	2	A7	STA	X	4	2
77	SRMA		7	3	91	CPA	S	4	2	A8	XRA	X	4	2
78	SLMA		7	3	92	SCA	S	4	2	A9	ACA	X	4	2

OP	MNEM	MODE	~	#	OP	MNEM	MODE	~	#	OP	MNEM	MODE	~	#
AA	ORA	X	4	2	C0	SUB	I	2	2	D9	ACB	S	4	2
AB	ADA	X	4	2	C1	CPB	I	2	2	DA	ORB	S	4	2
AC	CPX	X	6	2	C2	SCB	I	2	2	DB	ADB	S	4	2
AD	JS	X	7	2	C3	ADD	I	4	3	DC	LDD	S	5	2
AE	LDX	X	5	2	C4	ANB	I	2	2	DD	STD	S	5	2
AF	STX	X	5	2	C5	ANBT	I	2	2	DE	LDU	S	5	2
B0	SUA		5	3	C6	LDB	I	2	2	DF	STU	S	5	2
B1	CPA		5	3	C8	XRB	I	2	2	E0	SUB	X	4	2
B2	SCA		5	3	C9	ACB	I	2	2	E1	CPB	X	4	2
B3	SUD		7	3	CA	ORB	I	2	2	E2	SCB	X	4	2
B4	ANA		5	3	CB	ADB	I	2	2	E3	ADD	X	6	2
B5	ANAT		5	3	CC	LDD	I	3	3	E4	ANB	X	4	2
B6	LDA		5	3	CE	LDU	I	3	3	E5	ANBT	X	4	2
B7	STA		5	3	D0	SUB	S	4	2	E6	LDB	X	4	2
B8	XRA		5	3	D1	CPB	S	4	2	E7	STB	X	4	2
B9	ACA		5	3	D2	SCB	S	4	2	E8	XRB	X	4	2
BA	ORA		5	3	D3	ADD	S	6	2	E9	ACB	X	4	2
BB	ADA		5	3	D4	ANB	S	4	2	EA	ORB	X	4	2
BC	CPX		7	3	D5	ANBT	S	4	2	EB	ADB	X	4	2
BD	JS		8	3	D6	LDB	S	4	2	EC	LDD	X	5	2
BE	LDX		6	3	D7	STB	S	4	2	ED	STD	X	5	2
BF	STX		6	3	D8	XRB	S	4	2	EE	LDU	X	5	2

OP	MNEM	MODE	~	#	OP	MNEM	MODE	~	#	OP	MNEM	MODE	~	#
EF	STU	X	5	2	1026	JPRL NE		5(6)	4	10B3	CPD		8	4
F0	SUB		5	3	1027	JPRL EQ		5(6)	4	10BC	CPY		8	4
F1	CPB		5	3	1028	JPRL NV		5(6)	4	10BE	LDY		7	4
F2	SCB		5	3	1029	JPRL V		5(6)	4	10BF	STY		7	4
F3	ADD		7	3	102A	JPRL P		5(6)	4	10CE	LDS	I	4	4
F4	ANB		5	3	102B	JPRL N		5(6)	4	10DE	LDS	S	6	3
F5	ANBT		5	3	102C	JPRL GE		5(6)	4	10DF	STS	S	6	3
F6	LDB		5	3	102D	JPRL LT		5(6)	4	10EE	LDS	X	6	3
F7	STB		5	3	102E	JPRL GT		5(6)	4	10EF	STS	X	6	3
F8	XRB		5	3	102F	JPRL LE		5(6)	4	10FE	LDS		7	4
F9	ACB		5	3	103F	JI2		20	2	10FF	STS		7	4
FA	ORB		5	3	1083	CPD	I	5	4	113F	JI3		20	2
FB	ADB		5	3	108C	CPY	I	5	4	1183	CPU	I	5	4
FC	LDD		6	3	108E	LDY	I	4	4	118C	CPS	I	5	4
FD	STD		6	3	1093	CPD	S	7	3	1193	CPU	S	7	3
FE	LDU		6	3	109C	CPY	S	7	3	119C	CPS	S	7	3
FF	STU		6	3	109E	LDY	S	6	3	11A3	CPU	X	7	3
1021	JPR +2		5	4	109F	STY	S	6	3	11AC	CPS	X	7	3
1022	JPRL H		5(6)	4	10A3	CPD	X	7	3	11B3	CPU		8	4
1023	JPRL NH		5(6)	4	10AC	CPY	X	7	3	11BC	CPS		8	4
1024	JPRL NL		5(6)	4	10AE	LDY	X	6	3					
1025	JPRL C1		5(6)	4	10AF	STY	X	6	3					

~ Number of time states excluding post byte

 # Number of bytes excluding post byte

Figure B-11. Reverse Assembly

Type	Forms	Non Indirect				Indirect			
		Assembler Form	Postbyte OP Code	+ —	+ /	Assembler Form	Postbyte OP Code	+ —	+ /
Constant Offset From R (2's Complement Offsets)	No Offset	,R	1RR00100	0	0	(,R)	1RR10100	3	0
	5 Bit Offset	n,R	0RRnnnnn	1	0	defaults to 8-bit			
	8 Bit Offset	n, R	1RR01000	1	1	(n, R)	1RR11000	4	1
	16 Bit Offset	n, R	1RR01001	4	2	(n, R)	1RR11001	7	2
Accumulator Offset from R (2's Complement Offsets)	A Register Offset	A, R	1RR00110	1	0	(A, R)	1RR10110	4	0
	B Register Offset	B, R	1RR00101	1	0	(B, R)	1RR10101	4	0
	D Register Offset	D, R	1RR01011	4	0	(D, R)	1RR11011	7	0
Auto Increment/ Decrement R	Increment By 1	,R+	1RR00000	2	0	not allowed			
	Increment By 2	,R++	1RR00001	3	0	(,R++)	1RR10001	6	0
	Decrement By 1	,-R	1RR00010	2	0	not allowed			
	Decrement By 2	,--R	1RR00011	3	0	(,--R)	1RR10011	6	0
Constant Offsets From PC (2's Complement Offsets)	8 Bit Offset	n, PCR	1xx01100	1	1	(n, PCR)	1xx11100	4	1
	16 Bit Offset	n, PCR	1xx01101	5	2	(n, PCR)	1xx11101	8	2
Extended Indirect	16 Bit Address	—	—	—	—	(n)	10011111	5	2

R = X, Y, U or S
x = Don't Care

RR:
00=X
01=Y
10=U
11=S

$\pm$ and $\dagger$ indicate the number of additional cycles and bytes for the particular variation.

Figure B-12. Reverse Assembly, Indexed Addressing Modes

NON INDIRECT

Assembler Form R=X,Y,U,S	Post Byte Opcodes for R =			
	X	Y	U	S
,R+	80	A0	C0	E0
,R++	81	A1	C1	E1
,-R	82	A2	C2	E2
,—R	83	A3	C3	E3
,R	84	A4	C4	E4
B,R	85	A5	C5	E5
A,R	86	A6	C6	E6
n,R (8-bit)	88—	A8—	C8—	E8—
n,R (16-bit)	89——	A9——	C9——	E9——
D,R	8B	AB	CB	EB
[1]n,PC (8-bit)	8C—	AC—	CC—	EC—
[1]n,PC (16-bit)	8D——	AD——	CD——	ED——

INDIRECT

Assembler Form R=X,Y,U,S	Post Byte Opcodes for R =			
	X	Y	U	S
(,R+)	Not allowed			
(,R++)	91	B1	D1	F1
(,-R)	Not allowed			
(,—R)	93	B3	D3	F3
(,R)	94	B4	D4	F4
(B,R)	95	B5	D5	F5
(A,R)	96	B6	D6	F6
(n,R) (8-bit)	98—	B8—	D8—	F8—
(n,R) (16-bit)	99——	B9——	D9——	F9——
(D,R)	9B	BB	DB	FB
[1](n,PC) (8-Bit)	9C—	BC—	DC—	FC—
[1](n,PC) (16-bit)	9D——	BD——	DD——	FD——
(n)	9F——			

— = 8 bits of offset in the bytes immediately following the post byte.
[1] Register independent, all four forms are equivalent.

Figure B-13. Indexed Post Byte Permutations

Assembler Form R=X,Y,U,S	Post Byte Opcodes for R =			
	X	Y	U	S
15,R	0F	2F	4F	6F
14,R	0E	2E	4E	6E
13,R	0D	2D	4D	6D
12,R	0C	2C	4C	6C
11,R	0B	2B	4B	6B
10,R	0A	2A	4A	6A
9,R	09	29	49	69
8,R	08	28	48	68
7,R	07	27	47	67
6,R	06	26	46	66
5,R	05	25	45	65
4,R	04	24	44	64
3,R	03	23	43	63
2,R	02	22	42	62
1,R	01	21	41	61
-1,R	1F	3F	5F	7F
-2,R	1E	3E	5E	7E
-3,R	1D	3D	5D	7D
-4,R	1C	3C	5C	7C
-5,R	1B	3B	5B	7B
-6,R	1A	3A	5A	7A
-7,R	19	39	59	79
-8,R	18	38	58	78
-9,R	17	37	57	77
-10,R	16	36	56	76
-11,R	15	35	55	75
-12,R	14	34	54	74
-13,R	13	33	53	73
-14,R	12	32	52	72
-15,R	11	31	51	71
-16,R	10	30	50	70

Figure B-14. All Permutations of Non-Indirect 5-Bit Offset

SYMBOL	MEANING
←	Is transferred to
·	Boolean AND
+	Boolean OR
⊕	Boolean exclusive OR
(Overline)	Boolean NOT
:	Concatenation
+	Arithmetic plus
-	Arithmetic minus
×	Arithmetic Multiply

Figure B-15. Operating Notation

ABBREVIATION	MEANING
ACCA or A	Accumulator A
ACCB or B	Accumulator B
ACCA:ACCB or D	Double accumulator D
ACCX	Either accumulator A or B
E	Direct page register
EA	Effective address
F	Flag register
IFF	If and only if
IX or X	Index Register X
IY or Y	Index register Y
LSN	Least significant nibble
M	Memory Location
MI	Memory Immediate
MSN	Most significant nibble
PC	Program counter
R	A register before the operation
R'	A register after the operation
TEMP	Temporary storage location
xxH	Most significant byte of any 16-bit register
xxL	Least significant byte of any 16-bit register
Sp or S	Hardware Stack pointer
Us or U	User Stack pointer
P	A memory argument with Immediate, Short Direct, Direct, and Indexed addressing modes
Q	A read-modify-write argument with Short Direct, Indexed, and Direct addressing modes
( )	The data pointed to by the enclosed (16-bit address) or indirection
dd	8-bit jump offset
DDDD	16-bit jump offset

Figure B-16. Register Notation

AC **Add with Carry into Register** AC

Source Forms: ACA P; ACB P
Operation: R′ <— R + M + C
Flags:
D — Set if a half-carry is generated; cleared otherwise.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a carry is generated; cleared otherwise.
Description: Adds the contents of the C (carry) bit and the memory byte into an 8-bit accumulator.
Addressing Modes:
Immediate
Direct
Short Direct
Indexed

AD (8-Bit) **Add Memory into Register** AD (8-Bit)

Source Forms: ADA P; ADB P
Operation: R′ <— R + M
Flags:
D — Set if a half-carry is generated; cleared otherwise.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a carry is generated; cleared otherwise.
Description: Adds the memory byte into an 8-bit accumulator.
Addressing Modes:
Immediate
Direct
Short Direct
Indexed

AD (16-Bit) **Add Memory into Register** AD (16-Bit)

Source Forms: ADD P
Operation: R′ <— R + M:M + 1
Flags:
D — Not affected.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a carry is generated; cleared otherwise.
Description: Adds the 16-bit memory value into the 16-bit accumulator.
Addressing Modes:
Immediate
Direct
Short Direct
Indexed

ADX B **Add Accumulator B into Index Register X** ADX B

Source Forms: ADX B
Operation: IX′ <— IX + ACCB
Flags: Not affected.
Description: Adds the 8-bit unsigned value in accumulator B into index register X.
Addressing Modes: Inherent

AJAD **Adjust Register A Decimal** AJAD

Source Forms: AJAD

Operation: ACCA' <— ACCA + CF (MSN):CF(LSN)
where CF is a Correction Factor, as follows: the CF for each nibble (BCD) digit is determined separately, and is either 6 or 0.

Least Significant Nibble
CF(LSN) = 6 IFF 1) C = 1
or 2) LSN > 9

Most Significant Nibble
CF(MSN) = 6 IFF 1) C = 1
or 2) MSN > 9
or 3) MSN > 8 and LSN > 9

Flags:
D — Not affected.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Undefined.
C — Set if a carry is generated or if the carry bit was set before the operation; cleared otherwise.

Description: The sequence of a single byte add instruction on accumulator A (either ADA or ACA) and a following decimal addition adjust instruction results in a BCD addition with an appropriate carry bit. Both values to be added must be in proper BCD form (each nibble such that: $0 \leq$ nibble ≤ 9). Multiple-precision addition must add the carry generated by this decimal addition adjust into the next higher digit during the add operation (ACA) immediately prior to the next decimal addition adjust.

Addressing Modes: Inherent

AJW **Adjust Word to two Bytes** AJW

Source Forms: AJW

Operation: If bit seven of ACCB is set then ACCA' <— FF_{16}
else ACCA' <— 00_{16}

Flags:
D — Not affected.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Not affected.
C — Not affected.

Description: This instruction transforms a two's complement 8-bit value in accumulator B into a two's complement 16-bit value in the D accumulator.

Addressing Modes: Inherent

AN **Logical AND Memory into Register** AN

Source Forms: ANA P; ANB P

Operation: R' <— R • M

Flags:
D — Not affected.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.

Description: Performs the logical AND operation between the contents of an accumulator and the contents of memory location M and the result is stored in the accumulator.

Addressing Modes:
Immediate
Direct
Short Direct
Indexed

AN **Logical AND Immediate Memory into Flag (Condition Code) Register** AN

Source Forms: ANFI XX

Operation: R' <— R • MI

Flags: Affected according to the operation.

Description: Performs a logical AND between the flag register and the immediate byte specified in the instruction and places the result in the flag register.

Addressing Modes: Immediate

ANrT **And Bit Test** ANrT

Source Forms: ANAT P, ANBT P

Operation: TEMP <— R • M

Flags:
D — Not affected.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.

Description: Performs the logical AND of the contents of accumulator A or B and the contents of memory location M and modifies the flags accordingly. The contents of accumulator A or B and memory location M are not affected.

Addressing Modes:
Immediate
Direct
Short Direct
Indexed

CL **Clear** CL

Source Forms: CL Q

Operation:
TEMP <— M
M <— 00_{16}

Flags:
D — Not affected.
S — Always cleared.
Z — Always set.
V — Always cleared.
C — Always cleared.

Description: Accumulator A or B or memory location M is loaded with 00000000. Note that the EA is read during this operation.

Addressing Modes:
Inherent
Direct
Short Direct
Indexed

CMrA **Complement (Arithmetic)** CMrA

Source Forms: CMMA Q: CMAA; CMBA

Operation: M' <— 0 - M

Flags:
D — Undefined.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if the original operand was 10000000.
C — Set if a borrow is generated; cleared otherwise.

Description: Replaces the operand with its two's complement. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry. Note that 80_{16} is replaced by itself and only in this case is the V (overflow) bit set. The value 00_{16} is also replaced by itself, and only in this case is the C (carry) bit cleared.

Addressing Modes:
Inherent
Direct
Short Direct

CMrL **Complement (Logical)** CMrL

Source Forms: CMML Q; CMAL; CMBL

Operation: M' <— 0 + $\overline{M}$

Flags:
D — Not affected.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Always set.

Description: Replaces the contents of memory location M or accumulator A or B with its logical complement. When operating on unsigned values, only JPR EQ and JPR NE jumps can be expected to behave properly following a CMrL instruction: When operating on two's complement values, all signed jumps are available.

Addressing Modes:
Inherent
Direct
Short Direct
Indexed

CP (8-Bit) **Compare Memory from Register** CP (8-Bit)

Source Forms: CPA P; CPB P

Operation: TEMP <— R - M

Flags:
D — Undefined.
S — Set if the result if negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.

Description: Compares the contents of memory location to the contents of the specified register and sets the appropriate flags. Neither memory location M nor the specified register is modified. The carry flag represents a borrow and is set to the inverse of the resulting binary carry.

Addressing Modes:
Immediate
Direct
Short Direct
Indexed

CP (16-Bit) **Compare Memory from Register** CP (16-Bit)

Source Forms: CPD P; CPX P; CPU P; CPS P

Operation: TEMP < — R - M:M + 1

Flags:
D — Not affected.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.

Description: Compares the 16-bit contents of the concantenated memory locations M:M + 1 to the contents of the specified register and sets the appropriate flags. Neither the memory locations nor the specified register is modified unless autoincrement or autodecrement is used. The carry flag represents a borrow and is set to the inverse of the resulting binary carry.

Addressing Modes:
Immediate
Direct
Short Direct
Indexed

CPrZ **Compare with Zero** CPrZ

Source Forms: CPMZ Q; CPAZ; CPBZ

Operation: TEMP <—M - 0

Flags:
D — Not affected.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.

Description: Set the S (sign) and Z (zero) bits according to the contents of memory location M, and clear the V (overflow) bit. The CPrZ instruction provides only minimum information when testing unsigned values; since no unsigned value is less than zero, JPR L and JPR NH have no utility. While JPR H could be used after CPrZ, it provides exactly the same control as JPR NE, which is preferred. The signed jumps are available.

Addressing Modes:
Inherent
Direct
Short Direct
Indexed

Comments: The 6800 processor clears the C (carry) bit.

DC **Decrement** DC

Source Forms: DC Q; DCA; DCB

Operation: M' <— M - 1

Flags:
D — Not affected.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if the original operand was 10000000; cleared otherwise.
C — Not affected.

Description: Subtract one from the operand. The carry bit is not affected, thus allowing this instruction to be used as a loop counter in multiple-precision computations. When operating on unsigned values, only JPR EQ and JPR NE jumps can be expected to behave consistently. When operating on two's complement values, all signed jumps are available.

Addressing Modes:
Inherent
Direct
Short Direct
Indexed

HLT C **Halt, Clear Flag Bits and Wait for Interrupt** HLT C

Source Forms: HLT C XX

E	F	D	I	S	Z	V	C

Operation:

FR <— FR • MI (Possibly clear masks)
Set E (entire state saved)
SP' <— SP - 1, (SP) <— PCL
SP' <— SP - 1, (SP) <— PCH
SP' <— SP - 1, (SP) <— USL
SP' <— SP - 1, (SP) <— USH
SP' <— SP - 1, (SP) <— IYL
SP' <— SP - 1, (SP) <— IYH
SP' <— SP - 1, (SP) <— IXL
SP' <— SP - 1, (SP) <— IXH
SP' <— SP - 1, (SP) <— E
SP' <— SP - 1, (SP) <— ACCB
SP' <— SP - 1, (SP) <— ACCA
SP' <— SP - 1, (SP) <— F

Flags: Affected according to the operation.

Description: This instruction ANDs an immediate byte with the flag register which may clear the interrupt mask bits I and F, stacks the entire machine state on the hardware stack and then looks for an interrupt. When a non-masked interrupt occurs, no further machine state information need be saved before vectoring to the interrupt handling routine. This instruction replaced the 6800 CLI WAI sequence, but does not place the buses in a high-impedance state. A FIRQ (fast interrupt request) may enter its interrupt handler with its entire machine state saved. The RTI (return from interrupt) instruction will automatically return the entire machine state after testing the E (entire) bit of the recovered flag register.

Addressing Modes: Immediate

Comments: The following immediate values will have the following results:

FF = enable neither
EF = enable $\overline{\text{IRQ}}$
BF = enable $\overline{\text{FIRQ}}$
AF = enable both

HLT Z **Halt and Tri-State** HLT Z

Source Forms: HLT Z

Operation: Stop processing instructions.

Flags: Not affected

Description: When a HLT Z instruction is executed, the processor enters a synchronizing state, stops processing instructions, and waits for an interrupt. When an interrupt occurs, the synchronizing state is cleared and processing continues. If the interrupt is enabled, and it lasts three cycles or more, the processor will perform the interrupt routine. If the interrupt is masked or is shorter than three cycles, the processor simply continues to the next instruction. While in the synchronizing state, the address and data buses are in the high-impedance state.

This instruction provides software synchronization with a hardware process. Consider the following example for high-speed acquisition of data:

```
FAST  HLT      Z          Wait for data.
      Interrupt
      LDA      DISC       Data from disk and clear interrupt.
      STAX     ,X+        Put in Buffer.
      DCB                 Count it, done?
      JPR      NE FAST    Go again if not.
```

The synchronizing state is cleared by any interrupt. Of course, enabled interrupts at this point may destroy the data transfer and, as such, should represent only emergency conditions.

The same connection used for interrupt-driven I/O service may also be used for high-speed data transfers by setting the interrupt mask and using the HLT Z instruction as the above example demonstrates.

Addressing Modes: Inherent

IC **Increment** IC

Source Forms: IC Q; ICA; ICB

Operation: M′ <— M + 1

Flags:
D — Not affected.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if the original operand was 01111111; cleared otherwise.
C — Not affected.

Description: Adds to the operand. The carry bit is not affected, thus allowing this instruction to be used as a loop counter in multiple-precision computations. When operating on unsigned values, only the JPR EQ and JPR NE jumps can be expected to behave consistently. When operating on two's complement values, all signed jumps are correctly available.

Addressing Modes:
Inherent
Direct
Short Direct
Indexed

JI **Jump to Interrupt (Software Interrupt)** JI

Source Forms: JI

Operation:
Set E (entire state will be saved)
SP' <— SP - 1, (SP) <— PCL
SP' <— SP - 1, (SP) <— PCH
SP' <— SP - 1, (SP) <— USL
SP' <— SP - 1, (SP) <— USH
SP' <— SP - 1, (SP) <— IYL
SP' <— SP - 1, (SP) <— IYH
SP' <— SP - 1, (SP) <— IXL
SP' <— SP - 1, (SP) <— IXH
SP' <— SP - 1, (SP) <— E
SP' <— SP - 1, (SP) <— ACCB
SP' <— SP - 1, (SP) <— ACCA
SP' <— SP - 1, (SP) <— F
Set I, F (mask interrupts)
PC' <— (FFFA):(FFFB)

Flags: Not affected.

Description: All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself), and control is transferred through the software interrupt vector. Both the normal and fast interrupts are masked (disabled).

Addressing Modes: Inherent

JI2 **Jump to Interrupt 2 (Software Interrupt 2)** JI2

Source Forms: JI2

Operation:
Set E (entire state saved)
SP' <— SP - 1, (SP) <— PCL
SP' <— SP - 1, (SP) <— PCH
SP' <— SP - 1, (SP) <— USL
SP' <— SP - 1, (SP) <— USH
SP' <— SP - 1, (SP) <— IYL
SP' <— SP - 1, (SP) <— IYH
SP' <— SP - 1, (SP) <— IXL
SP' <— SP - 1, (SP) <— IXH
SP' <— SP - 1, (SP) <— E
SP' <— SP - 1, (SP) <— ACCB
SP' <— SP - 1, (SP) <— ACCA
SP' <— SP - 1, (SP) <— F
PC' <— (FFF4):(FFF5)

Flags: Not affected.

Description: All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself), and control is transferred through the software interrupt 2 vector. This interrupt is available to the end user and must not be used in packaged software. This interrupt does not mask (disable) the normal and fast interrupts.

Addressing Modes: Inherent

JI3 **Jump to Interrupt 3 (Software Interrupt 3)** JI3

Source Forms: JI 3

Operation:
Set E (entire state will be saved)
SP' <— SP - 1, (SP) <— PCL
SP' <— SP - 1, (SP) <— PCH
SP' <— SP - 1, (SP) <— USL
SP' <— SP - 1, (SP) <— USH
SP' <— SP - 1, (SP) <— IYL
SP' <— SP - 1, (SP) <— IYH
SP' <— SP - 1, (SP) <— IXL
SP' <— SP - 1, (SP) <— IXH
SP' <— SP - 1, (SP) <— E
SP' <— SP - 1, (SP) <— ACCB
SP' <— SP - 1, (SP) <— ACCA
SP' <— SP - 1, (SP) <— F
PC' <— (FFF2):(FFF3)

Flags: Not affected.

Description: All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack point itself), and control is transferred through the software interrupt 3 vector. This interrupt does not mask (disable) the normal and fast interrupts.

Addressing Modes: Inherent

JPR C0 **Jump Relative on Carry Clear** JPR C0

Source Forms: JPR C0 dd; JPRL C0 DDDD

Operation:
TEMP <— MI
IFF C = 0 then PC' <— PC + TEMP

Flags: Not affected.

Description: Tests the state of the C (carry) bit and causes a jump if it is clear.

Addressing Modes: Relative

Comments: Equivalent to JPR NL dd; JPRL NL DDDD

JPR C1 **Jump Relative on Carry Set** JPR C1

Source Forms: JPR C1 dd; JPRL C1 DDDD

Operation:
TEMP <— MI
IFF C = 1 then PC' <— PC + TEMP

Flags: Not affected.

Description: Tests the state of the C (carry) bit and causes a jump if it is set.

Addressing Modes: Relative

Comments: Equivalent to JPR L dd; JPRL L DDDD

JPR EQ **Jump Relative on Equal** JPR EQ

Source Forms: JPR EQ dd; JPRL EQ DDDD

Operation:
TEMP <— MI
IFF Z = 1 then PC' <— PC + TEMP

Flags: Not affected.

Description: Tests the state of the Z (zero) bit and causes a jump if it is set. When used after a subtract or compare operation, this instruction will jump if the compared values, signed or unsigned, were exactly the same.

Addressing Modes: Relative

JPR GE **Jump Relative on Greater than or Equal to Zero** JPR GE

Source Forms: JPR GE dd; JPRL GE DDDD

Operation: TEMP <— MI
IFF (S⊕V) = 0 then PC' <— PC + TEMP

Flags: Not affected.

Description: Causes a jump if the S (sign) bit and the V (overflow) bit are either both set or both clear. That is, a jump if the sign of a valid two's complement result is, or would be, positive. When used after a subtract or compare operation on two's complement values, this instruction will jump if the register was greater than or equal to the memory operand.

Addressing Modes: Relative

JPR GT **Jump Relative on Greater** JPR GT

Source Forms: JPR GT dd; JPRL GT DDDD

Operation: TEMP <— MI
IFF Z • (N⊕V) = 0 then PC' <— PC + TEMP

Flags: Not affected.

Description: Causes a jump if the S (sign) bit and the V (overflow) bit are either both set or both clear and the Z (zero) bit is clear. In other words, a jump if the sign of a valid two's complement result is, or would be, positive and not zero. When used after a subtract or compare operation on two's complement values, this instruction will jump if the register was greater than or equal to the memory operand.

Addressing Modes: Relative

JPR H **Jump Relative if Higher** JPR H

Source Forms: JPR H dd; JPRL H DDDD

Operation: TEMP <— MI
IFF (C + Z) = 0 then PC' <— PC + TEMP

Flags: Not affected.

Description: Causes a jump if the previous operation caused neither a carry nor a zero result. When used after a subtract or compare operation on unsigned binary values, this instruction will jump if the register was higher than the memory operand.

Addressing Modes: Relative

Comments: Generally not useful after IC/DC, LD/CPrZ, and CPrZ/CL/CMrL instructions.

JPR L **Jump Relative on Lower** JPR L

Source Forms: JPR L dd; JPRL L DDDD

Operation: TEMP <— MI
IFF C = 1 then PC' <— PC + TEMP

Flags: Not affected.

Description: Tests the state of the C (carry) bit and causes a jump if it is set. When used after a subtract or compare on unsigned binary values, this instruction will jump if the register was lower than the memory operand.

Addressing Modes: Relative

Comments: This is a duplicate assembly-language mnemonic for the single machine instruction JPR C1. Generally not useful after IC/DC, LD/ST, and CPrZ/CL/CMrL instructions.

JPR LE **Jump Relative on Less than or Equal to Zero** JPR LE

Source Forms: JPR LE dd; JPRL LE DDDD

Operation: TEMP <— MI
IFF Z + (S⊕V) = 1 then PC' <—PC + TEMP

Flags: Not affected.

Description: Causes a jump if the exclusive OR of the S (sign) and V (overflow) bits is 1 or if the Z (zero) bit is set. That is, a jump if the sign of a valid two's complement result is, or would be, negative. When used after a subtract or compare operation on two's complement values, this instruction will jump if the register was less than or equal to the memory operand.

Addressing Modes: Relative

JPR LT **Jump Relative on Less than Zero** JPR LT

Source Forms: JPR LT dd; JPRL LT DDDD

Operation: TEMP <— MI
IFF (S⊕V) = 1 then PC' <—PC + TEMP

Flags: Not affected.

Description: Causes a jump if either, but not both, of the S (sign) or V (overflow) bits is set. That is, jump if the sign of a valid two's complement result is, or would be, negative. When used after a subtract or compare operation on two's complement binary values, this instruction will jump if the register was less than the memory operand.

Addressing Modes: Relative

JPR N **Jump Relative on Negative**

Source Forms: JPR N dd; JPRL N DDDD

Operation: TEMP <— MI
IFF S = 1 then PC' <— PC + TEMP

Flags: Not affected.

Description: Tests the state of the S (sign) bit and causes a jump if set. That is, a jump if the sign of the two's complement result is negative.

Addressing Modes: Relative

Comments: When used after an operation on signed binary values, this instruction will jump if the result is minus. It is generally preferred to use the JPRL LT instruction after signed operations.

JPR NE **Jump Relative Not Equal** JPR NE

Source Forms: JPR NE dd; JPRL NE DDDD

Operation: TEMP <— MI
IFF Z = 0 then PC' <— PC + TEMP

Flags: Not affected.

Description: Tests the state of the Z (zero) bit and causes a jump if it is clear. When used after a subtract or compare operation on any binary values, this instruction will jump if the register is, or would be, not equal to the memory operand.

Addressing Modes: Relative

JPR NH **Jump Relative on Not Higher** JPR NH

Source Forms: JPR NH dd; JPRL NH DDDD

Operation: TEMP <— MI
IFF (C + Z) = 1 then PC' <— PC + TEMP

Flags: Not affected.

Description: Causes a jump if the previous operation caused either a carry or a zero result. When used after a subtract or compare operation on unsigned binary values, this instruction will jump if the register was lower than or the same as the memory operand.

Addressing Modes: Relative

Comments: Generally not useful after IC/DC, LD/ST, and CPrZ/CL/CMrL instructions.

JPR NL **Jump Relative if Not Lower** JPR NL

Source Forms: JPR NL dd; JPRL NL DDDD

Operation: TEMP <— MI
IFF C = 0 then PC' <— PC + MI

Flags: Not affected.

Description: Tests the state of the C (carry) bit and causes a jump if it is clear. When used after a subtract or compare on unsigned binary values, this instruction will jump if the register was higher than or the same as the memory operand.

Addressing Modes: Relative

Comments: This is a duplicate assembly-language mnemonic for the single machine instruction JPR C0. Generally not useful after IC/DC, LD/ST, and CPrZ/CL/CMrL instructions.

JPR NV **Jump Relative on Overflow Clear** JPR NV

Source Forms: JPR NV dd; JPRL NV DDDD

Operation: TEMP <— MI
IFF V = 0 then PC' <— PC + TEMP

Flags: Not affected.

Description: Tests the state of the V (overflow) bit and causes a jump if it is clear. That is, a jump if the two's complement result was valid. When used after an operation on two's complement binary values, this instruction will jump if there was no overflow.

Addressing Modes: Relative

JPR P **Jump Relative on Positive** JPR P

Source Forms: JPR P dd; JPRL P DDDD

Operation: TEMP <— MI
IFF S = 0 then PC' <— PC + TEMP

Flags: Not affected.

Description: Tests the state of the S (sign) bit and causes a jump if it is clear. That is, jump if the sign of the two's complement result is positive.

Addressing Modes: Relative

Comments: When used after an operation on signed binary values, this instruction will jump if the result (possibly invalid) is positive. It is generally preferred to use the JPR GE instruction after signed operations.

JPR UN **Jump Relative Unconditionally** JPR UN

Source Forms: JPR UN dd; JPRL UN DDDD
Operation: TEMP <— MI
PC' <— PC + TEMP
Flags: Not affected.
Description: Causes an unconditional jump.
Addressing Modes: Relative

JPR V **Jump Relative on Overflow Set** JPR V

Source Forms: JPR V dd; JPRL V DDDD
Operation: TEMP <— MI
IFF V = 1 then PC' <— PC + TEMP
Flags: Not affected.
Description: Tests the state of the V (overflow) bit and causes a jump if it is set. That is, a jump if the two's complement result was invalid. When used after an operation on two's complement binary values, this instruction will jump if there was an overflow.
Addressing Modes: Relative

JPR + n **Jump Never (Skip n)** JPR + n

Source Forms: JPR + 1 dd; JPR + 2 DDDD
Operation: TEMP <— MI
Flags: Not affected.
Description: Does not cause a jump. This instruction is essentially a no operation, but has a bit pattern logically related to jump relative unconditionally.
Addressing Modes: Relative

JP UN **Jump Unconditionally** JP UN

Source Forms: JPR UN EA
Operation: PC' <— EA
Flags: Not affected.
Description: Program control is transferred to the effective address.
Addressing Modes: Direct
Short Direct
Indexed

JS UN **Jump to Subroutine** JS UN

Source Forms: JS UN EA
Operation: SP' <— SP - 1, (SP) <— PCL
SP' <— SP - 1, (SP) <— PCH
PC' <— EA
Flags: Not affected.
Description: Program control is transferred to the effective address after storing the return address on the hardware stack. An RTS instruction should be the last executed instruction of the subroutine.
Addressing Modes: Direct
Short Direct
Indexed

JSR **Jump Relative to Subroutine** JSR

Source Forms: JSR dd; JSRL DDDD

Operation:
TEMP <— MI
SP' <— SP - 1, (SP) <— PCL
SP' <— SP - 1, (SP) <— PCH
PC' <— PC + TEMP

Flags: Not affected.

Description: The program counter is pushed onto the stack, and then loaded with the sum of the program counter and the offset.

Addressing Modes: Relative

Comments: A return from subroutine (RTS) instruction is used to reverse this process and must be the last instruction executed in a subroutine.

LD (8-Bit) **Load Register from Memory** LD (8-Bit)

Source Forms: LDA P; LDB P

Operation: R' <— M

Flags:
D — Not affected.
S — Set if the loaded data is negative; cleared otherwise.
Z — Set if the loaded data is zero; cleared otherwise.
V — Always cleared.
C — Not affected.

Description: Loads the contents of memory location M into the designated register.

Addressing Modes:
Immediate
Direct
Short Direct
Indexed

Subtract with Borrow — SC

SCA P; SCB P

R′ <— R - M - C

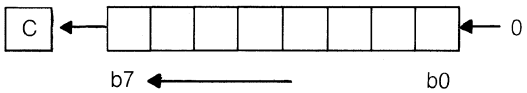
D — Undefined.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise
C — Set if a borrow is generated; cleared otherwise.

Subtracts the contents of memory location M and the borrow (in the C [carry] bit) from the contents of the designated 8-bit register, and places the result in that register. The C bit represents a borrow and is set to the inverse of the resulting binary carry.

s: Immediate
Direct
Short Direct
Indexed

Shift Left Arithmetic — SLrA

SLMA Q; SLAA; SLBA



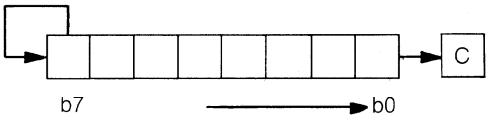
D — Undefined.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Loaded with the result of the exclusive OR of bits six and seven of the original operand.
C — Loaded with bit seven of the original operand.

Shifts all bits of the operand one place to the left. Bit zero is loaded with a zero. Bit seven is shifted into the C (carry) bit.

s: Inherent
Direct
Short Direct
Indexed

Shift Right Arithmetic — SRrA

SRMA Q; SRAA; SRBA



D — Undefined.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Not affected.
C — Loaded with bit zero of the original operand.

Shifts all bits of the operand one place to the right. Bit seven is held constant. Bit zero is shifted into the C (carry) bit.

s: Inherent
Direct
Short Direct
Indexed

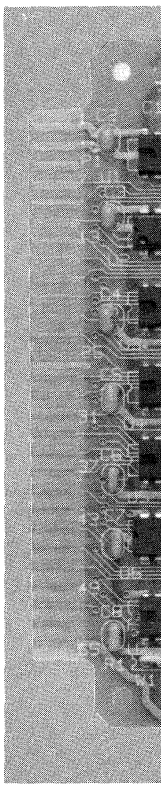
PURPOSE

The 7809 is
card with onboa
port. The card
operates at .921
MHz.

The serial
baud rates from
communication
equipment (DT
programmed fo
and modem co

The memor
sockets. These
byte-wide RAMs

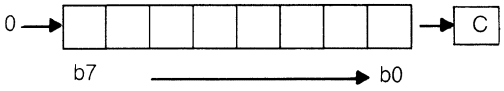
The card als
latches the co
acknowledge. T
feature allows
cards that gener



SRrL **Shift Right Logical** SRrL

Source Forms: SRML Q; SRAL; SRBL

Operation:



Flags:
D — Not affected.
S — Always cleared.
Z — Set if the result is zero; cleared otherwise.
V — Not affected.
C — Loaded with bit zero of the original operand.

Description: Performs a logical shift right on the operand. Shifts a zero into bit seven and bit zero into the C (carry) bit.

Addressing Modes:
Inherent
Direct
Short Direct
Indexed

ST (8-bit) **Store Register into Memory** ST (8-bit)

Source Forms: STA P; STB P

Operation: M' <— R

Flags:
D — Not affected.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.

Description: Writes the contents of an 8-bit register into a memory location.

Addressing Modes:
Direct
Short Direct
Indexed

ST (16-Bit) **Store Register into Memory** ST (16-bit)

Source Forms: STD P; STX P; STY P; STS P; STU P

Operation: M':M + 1' <— R

Flags:
D — Not affected.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.

Description: Writes the contents of an 16-bit register into two consecutive memory locations.

Addressing Modes:
Direct
Short Direct
Indexed

SU (8-Bit) **Subtract Memory from Register** SU (8-Bit)

Source Forms: SUA P; SUB P

Operation: R' <— R - M

Flags:
D — Undefined.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if the overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.

Description: Subtracts the value in memory location M from the contents of a designated 8-bit register. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry.

Addressing Modes:
Inherent
Direct
Short Direct
Indexed

SU (16-Bit) **Subtract Memory from Register** SU (16-Bit)

Source Forms: SUD P

Operation: R' <—R - M:M + 1

Flags:
D — Not affected.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if the overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.

Description: Subtracts the value in memory location M:M + 1 from the contents of a designated 16-bit register. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry.

Addressing Modes:
Immediate
Direct
Short Direct
Indexed

D — Not affected.

XCR **Exchange Registers** XCR

Source Forms: XCR R1,R2

Operation: R1 <—> R2

Flags: Not affected (unless one of the registers is the flag register).

Description: Exchanges data between two designated registers. Bits 3-0 of the post byte define one register, while bits 7-4 define the other, as follows:

0000 = A:B	1000 = A
0001 = X	1001 = B
0010 = Y	1010 = F
0011 = US	1011 = E
0100 = SP	1100 = Undefined
0101 = PC	1101 = Undefined
0110 = Undefined	1110 = Undefined
0111 = Undefined	1111 = Undefined

Only like-size registers may be exchanged. (8-bit with 8-bit or 16-bit with 16-bit).

Addressing Modes: Immediate

XR **Exclusive OR** XR

Source Forms: XRA P; XRB P

Operation: R' <— R⊕M

Flags:
D — Not affected.
S — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.

Description: The contents of memory location M is exclusive ORed into an 8-bit register.

Addressing Modes:
Immediate
Direct
Short Direct
Indexed

PRO-LOG® CORPORATION

Yes, I'm interested in Pro-Log.

Please send me more information about your:

- ☐ 7000 Series STD BUS Cards
- ☐ STD BUS Packaging Solutions
- ☐ STD BUS Subsystem
- ☐ STD BUS Systems
- ☐ Designer Software
- ☐ Design Courses and Seminars
- ☐ Programmer Product Line
- ☐ Other _____

- ☐ I am currently using Pro-Log equipment.
- ☐ Add my name to your mailing list.
- ☐ Please have a Sales Representative contact me.

FOLD

Cut Here

FOLD

Reader Comments: _____

Name _____ City _____

Company _____ State _____ Zip _____

Address _____ Telephone _____

PLACE
POSTAGE
HERE

PRO-LOG®
CORPORATION
2411 GARDEN ROAD
MONTEREY, CA 93940

Attn: Literature Department